MODEL-BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES

THESIS

Ralph W. Dries
Flight Lieutenant, RAAF

AFIT/GSO/ENG/90D-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSO/ENG/90D-03

(1)

MODEL-BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES

THESIS

Ralph W. Dries
Flight Lieutenant, RAAF

AFIT/GSO/ENG/90D-03

Approved for public release; distribution unlimited.

THESIS APPROVAL

STUDENT: Ralph W. Dries **CLASS:** GSO-90D

THESIS TITLE: Model-Based Reasoning in the Detection of Satellite Anomalies

DEFENSE DATE: Monday, November 26, 1990

COMMITTEE:	**NAME/DEPARTMENT**	**SIGNATURE**
Advisor/Co-Advisor (circle appropriate role)	Dr F.M. Brown, ENG	F.M. Brown
Co-Advisor/ENS Representative (circle appropriate role)	Maj B. Morlan, ENS	B. L. Morlan
Reader	_____	_____

AFIT/GSO/ENG/90D-03

MODEL-BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Space Operations

Ralph W. Dries, B.E.

Flight Lieutenant, RAAF

December, 1990

Preface

The purpose of this thesis was for me to learn something about satellites, artificial intelligence (AI), and programming using an AI language. I got more than I had bargained for! All of the above were achieved. Although I now know a great deal more than I knew before, the research effort has made me aware of how little I really do know. The subject matter is huge, and shuffling through the copious research papers was no easy task. I wish to thank my thesis adviser, Dr. Frank Brown, for his support and encouragement. He always had a knack of improving my glum disposition during the course of our meetings. I would also like to thank Captain Jim Skinner of the Advanced Technology Center at Kirkland AFB, and Captain Jeff Carstens of DSCS SPO in Los Angeles, who were instrumental in providing the satellite orbital operations handbook that provided important data for this thesis. Finally, I wish to make a special thanks to my wife, Jennifer, and children, Travis (7), David (5), Danielle (3), and Benjamin (1), who have had to do without a husband and father for so long.

Ralph W. Dries

Table of Contents

List of Figures

List of Tables

Abstract

Automatic fault detection and recovery would be a mandatory requirement for a satellite where some degree of autonomy is required. This thesis reviews some AI techniques used for the detection of satellite anomalies, and concludes that the model-based reasoning paradigm is best suited for automated on-board fault detection because it can cope with situations not necessarily programmed into the knowledge base. Using the Scheme language (a dialect of LISP) and its SCOOPS object oriented extension, development of software is described that models the pitch control channel in the attitude and velocity control subsystem of a typical geo-stationary communications satellite. This model is used by the model-based reasoning algorithm to diagnose faults in the real system. The algorithm used, is based on Scarl's "Full Consistency Algorithm", which is suitable for systems that have many sensors, but has limitations when applied to systems that are dependent on time or have feedback loops. These limitations were overcome by using a model that did not include time dependent objects and by "breaking the loop". It was found, for this problem domain, that the reasoner's model did not have to be identical to the real system to be able to successfully detect the cause of an anomaly.

MODEL-BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES

I. Introduction

This chapter provides some background to the satellite anomaly detection problem. It states the problem and the objective of this thesis work, and gives an overview of the methodology used to solve the problem and achieve the objective.

Background

Autonomous satellite operation has been an endeavor of spacecraft designers ever since the first satellite launches in the mid-1950s. Satellite operation is a complex and people-consuming business and even now, in the 1990s, is still very dependent on ground support. Autonomous operation is a matter of survival. In times of conflict, vulnerable ground stations will become targets, and if destroyed, hundreds of satellites can fly aimlessly out of control. Autonomous operation is also a matter of removing the burden of complex operation from the people. The people who "fly" these satellites are experts in their field and take years of training and experience to become effective. With Shuttle operations back in full swing, the number of satellites is increasing and with the introduction of the Strategic Defence Initiative (SDI), the numbers can become so great that there may not be enough human experts to control these spacecraft or ground systems to support them.

The "experts" are in greatest demand when things go wrong and the anomalous situation has to oe quickly resolved. Owing to Artificial Intelligence (AI) techniques developed over recent years, "not-so-expert" operators can resolve

anomalous situations by taking advice from a rule-based expert system. Although these expert systems alleviate the problem of having many experts on hand, *rule-based* expert systems suffer from a major drawback: they reason using heuristics. Heuristics are the "rules-of-thumb" given to the knowledge engineer by the expert. Because of system complexity, heuristic reasoning (often called shallow reasoning) cannot account for every possible anomalous situation that may occur. This deficiency has two consequences. First, experts will still be required to solve anomalies not accounted for in the rule base, and second, the expert system cannot reliably be included as an onboard system, and so cannot be used to make the satellite truly autonomous.

If true satellite autonomy is ever to be realized in the domain of anomaly detection, a better method than rule-based expert systems must be found. Model-based reasoning (also known as model-based expert system) may be that method. Model-based reasoning uses a computer model of the system running in parallel with the real system. The continuous comparison of sensor data and the states in the computer model generates a discrepancy signal if an anomaly occurs. The anomaly is then resolved by a deeper reasoning process that depends only on the system's structure and function, and not on heuristics. Provided the real system can be accurately modelled and there exist sufficient sensors, this method promises to resolve anomalous situations not considered in the design of the model (Fulton, 1990:55).

Problem and Objective

The problem is that current rule-based expert systems reason using heuristics and cannot account for every anomaly that may occur in a complex satellite system. The objective of this thesis is to investigate the application of model-based reasoning to satellite anomaly detection. This will be done by building a prototype model-based

expert system that can be validated against real satellite telemetry data or computer simulation of a satellite system.

Scope

To make a satellite *totally* autonomous requires the automation of *all* aspects of satellite operation. This study focuses on the fault-management aspect of satellite operations, i.e., anomaly detection. The model-based expert system prototype will be ground-based and applied only to a subsystem of the satellite.

Methodology

The following methodology briefly outlines procedures used to attain the objective listed above:

Understanding the Satellite Anomaly Detection Problem. An appreciation of the problems associated with satellite anomaly detection was achieved by searching the literature on this subject. First, a search on manual methods currently used to diagnose faults on satellites was made. Second, various AI techniques used to achieve the same result was studied in the literature. This study, recorded in Chapter II of this thesis, provided a good foundation and understanding of the anomaly detection problem.

Understanding the Model-Based Reasoning Paradigm. To provide an understanding of how model-based reasoning is used in fault diagnosis, research was carried out by reviewing literature, mostly research papers, on the subject. Particular attention was paid to techniques relating to anomaly detection in spacecraft, and the problems associated with that domain. The research in this area is recorded in Chapter III.

Determination of the Satellite Subsystem to Model. Detailed information required to characterize satellite subsystems was studied. This information came from literature search, satellite manuals and past AFIT theses that have used rule-based expert systems or other methods to conduct anomaly detection. The most useful document for this research came from an Orbital Operations Handbook for a geo-stationary satellite. The subsystem selected was the pitch control channel of the attitude and velocity control subsystem Details on this subsystem have been recorded in Chapter IV.

Finding the Best Computer Language to Use. To find the best computer language for this study, a review was made of several AI sybolic languages, expert system shells and object-oriented languages. Information for this came from literature search and through interviews with people experienced with the languages in question. Out of a short list of four languages suitable, and available, the language Scheme with the SCOOPS object oriented extension was selected as the language for prototyping the system. Selection criteria and other information on this language has been recorded in Chapter V.

Development of the Prototype Software. Software was developed that implements one of the model-based reasoning paradigms examined in Chapter III. Software was also developed that simulates the pitch control channel in the Attitude Control system. This model was used by the reasoner, and a copy was used to simulate the real pitch control channel to test the reasoner software. The software development is recorded in Chapter V.

Documentation of Results and Analysis. A number of tests were carried out on the designed model, and the diagnostic reasoner. Results and analysis of these tests is recorded in Chapter VI. Validation of the software was carried out using a computer simulation of the pitch control channel.

Drawings Conclusions and Making Recommendations. Conclusions were drawn and recommendations were made on the basis this thesis research, and from the results and analysis of the developed prototype model-based reasoner. Conclusions and recommendations are recorded in Chapter VII.

II. The Detection of Satellite Anomalies

Introduction

The review of literature presented here is the first of two parts. The first part is intended to provide an understanding of the satellite anomaly detection problem and was done by a literature review of the current methods used to diagnose anomalies in satellites. This chapter includes a brief look at the satellite systems and subsystems themselves, the way anomaly detection is done manually, and a study of some AI techniques used to get the same results. The second part of the literature review (next chapter) looks deeper into the various model-based reasoning techniques used in many systems ranging from jet aircraft to electronic circuits. The background research from these two studies should provides a foundation for the design of an *expert system* to diagnose satellite anomalies using model-based reasoning.

To gain an understanding of the anomaly detection problem, this chapter's main aim is to review current AI techniques used in the detection of satellite anomalies. This is done by describing a typical satellite system and highlighting the subsystems where detection of anomalies is frequently required. Next, a brief description of manual methods for anomaly detection, as used in a ground control environment, is given. This is followed by examples of various AI techniques that are currently used to diagnose faults in satellite systems.

Satellite Description

In the context of this paper "satellite" refers to any craft operating outside the earth's atmosphere. That includes craft that are manned or unmanned, in earth orbit or undergoing inter-planetary travel. Most satellites (estimated at about 95%) are

unmanned and in earth orbit; the remainder include such craft as the space shuttle, interplanetary craft (such as Voyager and Galileo), and future craft such as the National Aero Space Plane (NASP) and the Space Station. Future spacecraft are expected to be so complex that humans will not be capable of controlling the total systems without the aid of AI techniques.

Following is a description of the major subsystems in a satellite:

Electrical Power Subsystem. The electrical power subsystem ensures operation of the satellite over its lifetime. The power source can be electro-chemical (batteries, fuel cells etc.), solar (most common), or nuclear (usually radio isotope).

Guidance and Control Subsystem. The guidance and control subsystem ensures that the satellite is in the correct orbit and pointing in the right direction.

Propulsion Subsystem. Propulsion is obviously required for interplanetary missions, the space shuttle and the NASP, but less obvious is the requirement for fuel on-board an unmanned earth-orbiting satellite. Orbital perturbation due to the oblateness of the earth, third-body effects (sun, moon and planets) and drag cause satellites to be shifted from their nominal orbits, hence propulsion is required to correct for this defect (Cochran, 1985:Ch 2, 41-44). This is especially true for geo-stationary satellites which must remain fixed at a certain point above the earth. When the fuel runs out, these satellites are out of control.

Mission Requirements Subsystem. This subsystem contains the hardware, such as sensors and scientific instruments, required to achieve the mission of the satellite.

Communications, Command and Control (C^3) Subsystem. Today's satellites, even manned craft, are controlled by ground stations. The C^3 subsystem includes all the hardware and software necessary to ensure complete control of the satellite. It includes the transmitters, receivers, antennas, computers, telemetry hardware (that

relay the status of on-board sensors to the ground) and transducers that convert command signals from the ground to electrical and mechanical operations that control the satellite. Maintaining 24-hour communications with the satellite is a significant operation in itself. Communication can only occur when the satellite is in view, necessitating the use of worldwide tracking and commanding facilities (Barry, 1987:276).

Environmental Subsystem. The environmental subsystem of a *manned* spacecraft is a significant part of the overall system. Even for smaller unmanned spacecraft, this subsystem is required to monitor and adjust to environmental effects such as overheating (due to sunlight and internally generated heat), space charging effects (due to energetic space radiation and flying through charged particles) and space debris (which includes micrometeors) (Koons and Gorney, 1988:1).

Anomalies in any of the above subsystems can spell the end of a mission. Historically, it has been found that two subsystems in particular, the electrical subsystem and the guidance and control subsystem, have a much higher probability of breaking down than do the others. These subsystems are clearly defined and have an order of complexity that make them good candidates for automation of fault diagnosis and recovery (Passani and Brindle, 1986:255; Rampino, 1987:4; Howlin and others, 1988:178). This is supported by the relatively large number of expert systems that use these subsystems as their domain for anomaly detection.

Manual Methods of Anomaly Detection

In his discussion on the present satellite command and control environment, Barry stated that every hour of satellite operation requires hundreds of man-hours of ground control (Barry, 1987:276). This statement gives a clear indication of the hours of work required to sustain satellite operations. It also supports the growing

interest and pressing need for using AI techniques to ensure operability when the number of satellites becomes large and their systems more complex. For the domain of anomaly detection, people also can become a critical resource as a team of satellite engineers is usually required to resolve non-routine anomalous situations. This is even more prevalent in the US Air Force, which has total control of the NAVSTAR Global Positioning System (GPS), because their satellite engineers are largely military personnel who are transferred to other assignments just when their expertise is peaking (Rampino, 1987:2).

Received telemetry data that is stored on disk or tapes is the basis of present diagnostic techniques. Sensor status is extracted from this digital data stream and continuously monitored to determine the health-state of the satellite. Decoded information from the tapes is examined when an anomalous situation occurs and the symptoms compared to a hierarchical fault tree in an operations manual. The problem with this deterministic-check-list approach is that the data points must be prespecified to isolate the problem (Barry, 1987:278).

When the cause cannot be determined from the fault tree, the data is analyzed by the experts. Computers sometimes aid in this analysis but only to the extent of performing trend analysis on the telemetry data. This manual (or computer-aided manual) way of doing business is very time consuming and in a critical or military situation may not be acceptable.

On fault confirmation, operators effect recovery by issuing telecommands to switch in redundant systems; alternatively, they may reconfigure a subsystem and live with a degraded system. Usually, the first step taken is "safing" the satellite by removing power from suspected units to prevent further damage or possible loss of the spacecraft.

AI Techniques for Anomaly Detection

Rule-Based Expert Systems

The ground control environment was one of the first areas that saw the use of AI techniques in the field of anomaly detection. Several rule-based expert system prototypes have been designed that use the information from the fault tree in the operations manual and the expertise captured from the satellite engineers. The expert system is used strictly in an advisory capacity. It responds to an operator's input (fault symptoms) by generating a list of probable causes, usually one or more suggested solutions, and if asked, the reasoning used to come up with the result.

A rule-based expert system is an AI technique that uses knowledge in the form of heuristic rules (if ... then...) to infer a conclusion given the facts. The rule-based expert system is probably the most mature of the AI techniques used to date. This is evidenced by the comparatively larger number (compared with other AI techniques for fault detection) of prototypes that have been designed for anomaly detection in recent years. Many have been successfully tested as prototypes, but fully operational expert systems are rare in the literature.

The following examples are representative of the rule-based expert systems that exist as prototypes today:

NAVARES. The NAVstar Anomaly Resolution Expert System prototype is specifically designed for inexperienced US Air Force satellite operators who control the Global Positioning System (GPS) satellites (21 satellites in the mature system) without extensive contractor support. It has been tested and successfully diagnoses many anomalies in the Attitude, Velocity and Control Subsystem, and the Electrical Power Subsystem. Users interact with NAVARES by answering queries about the status of the subsystem components. NAVARES then uses its expert knowledge to

diagnose the anomaly and recommend a remedy. NAVARES uses the GURU expert system tool and database manager (Rampino, 1987).

MOORE. MOORE is another rule-based expert system prototype; it gets its name from the expert, Mr Robert J. Moore, who provided the knowledge. Its function is to assist in the diagnosis of problems with the antenna-pointing and earth-pointing functions of the attitude control system on the Tracking and Data Relay satellite (TDRS). TDRS is an operational geosynchronous communications satellite. Like NAVARES, MOORE is intended to assist operators at the TDRS ground terminal in troubleshooting problems that are not readily solved with routine procedures, and without expert counsel. MOORE uses Texas Instruments' Personal Consultant Plus expert system building tool. It was selected over Teknowledge's M.1 because of smoother graphics integration, superior editing capability, preferable development environment, and attractive local support and documentation (Howlin and others, 1988).

DAM. The Data Analysis Module is the first of four expert systems being developed by the Jet Propulsion Laboratory for a prototype Generic Payload Operations Control System (GPOCS). Unlike NAVARES and MOORE, which query the operator about the status of satellite subsystem components, DAM monitors incoming telemetry data and performs trend analysis based on a knowledge base and historic data archived on an optical disk storage device. The aim of the system is to respond very quickly to a developing problem, isolate and diagnose faults, pinpoint the probable fault location and recommend corrective action. DAM uses the CLIPS forward chaining expert system shell (Busse, 1988).

PMAD. The Power Management And Distribution system is a combination expert system and graphics interface prototype that allows an astronaut on board the space station to diagnose problems with the power subsystem. The astronaut enters sensor status on a graphics workstation and the underlying expert system (transparent

to the user) determines a probable cause using its rule-base. The suspect component is flashed in red on a schematic of the system under test. Although the prototype does not have enough rules to diagnose anomalies in a real space station, it is an excellent vehicle for expert system development. Indeed, one of its aims was to serve as a training tool for potential users and for those designing the expert system and developing the knowledge-base. The system is build on a Symbolics 3675 LISP-machine running the KEE expert system shell and the IRIS 3120 graphics workstation. The graphics interface and the communications interface (between the Symbolics and IRIS) are written in C (Hester, 1988).

All the above rule-based expert systems are used strictly in an advisory capacity. That is, the human has the final say. Graphics also play an important role in the human/computer interface. MOORE, DAM and PMAD have placed a high priority on color-graphics interfaces to ensure efficient operation and ease of data interpretation by the operator.

Although these expert systems alleviate the problem of having many experts on hand, *rule-based* expert systems suffer from a major drawback: they reason using heuristics. Heuristics are the "rules-of-thumb" given to the knowledge engineer by the expert. Unfortunately, heuristic reasoning (often called shallow reasoning) cannot account for every possible anomalous situation that may occur in a system as complex as a satellite (Cochran, 1985:49). This deficiency has two consequences. First, a demand for human experts will still exist to solve anomalies not accounted for in the rule base, and second, a satellite cannot be made truly autonomous using an expert system that does not reliably cover all failure modes.

Ease of programming and rapid prototyping is the greatest advantage of rule-based expert systems. However, if true satellite autonomy is ever to be realized in the domain of anomaly detection, better methods than rule-based expert systems must be found.

Causal Networks

A causal-network expert system attempts to overcome some of the problems associated with rule-based expert systems. Rule-based expert systems lack causal knowledge because they do not have an understanding of the underlying causes and effects in a system (Giarratano, 1989:8).

A causal-knowledge expert system is constructed by using a network of nodes and arcs to model the cause-effect relationships of the system in question. Consider two nodes A and B, and an arc from A to B. The A node is a symptom (failure state) and the B node is a cause of that symptom. The arc usually has a certainty-factor (CF) weighting between 0 and 1 that provides a measure of the degree to which B is a cause of A. Human experts determine the structure of the network and the weight values on each arc.

During a typical diagnosis run, fault hypothesis testing is carried out on each node on a path from an input symptom to an output cause. Depending on the outcome of the tests, an assignment of a CF value (initially set to zero) is made to each node. The inference engine cycles through three steps: selection and execution of the most appropriate test; analysis of the test result; and, modification of the CF of the nodes. When testing is complete on all path combinations, the terminal node with the highest certainty factor gives the most probable cause. The path used to reach that node can also be displayed for justification of the reasoning process (Longoni and others, 1987).

Multiple inputs and outputs cause the network to become complicated quickly. For example, several arcs with different weights may exit from one node, showing that several causes may account for one symptom. Also, several arcs may enter one node, showing that several symptoms may result from one cause. To complicate

matters even further, the network is usually more than one layer deep. The internal nodes represent symptoms that may be caused by other failure states.

Following is an example using a causal network expert for detection of satellite anomalies:

CANDIES. The CAusal Network DIagnostic Expert System is a conceptual prototype for the diagnosis of the Data Management Subsystem on a satellite. This system differs from the others in that it is intended as an *on-board* expert system with emphasis on autonomy. For an on-board system, Longoni states that expert system should provide facilities to:

> ...handle emergency situations when there is a loss of ground contact; enhance the automatic operation for deep space vehicles where reaction time due to long distances could be longer than needed; resolve the problems encountered because of minimal ground contact for low earth orbit spacecraft; and, reduce the costly involvement of human operators. (Longoni and others, 1987:294)

CANDIES will have a distributed architecture in which a supervisor expert system manages the communication among several expert subsystems. Each expert subsystem will monitor, diagnose and plan recovery action for the subsystem it controls in real time. For the first implementation, the Data Management Subsystem (DMSS) was chosen because: it is important to the rest of the spacecraft; much local knowledge is available about the subsystem; and, fault diagnosis on that subsystem promises effective results in the near term. The prototype is able to deal with 17 symptoms, selects up to 56 tests to end up with 40 different diagnosed faults. The CANDIES prototype is run on a VAX and is written in Franz LISP (Longoni and others, 1987).

The biggest disadvantage of causal-network reasoning is the requirement to account for almost every possible symptom and its associated cause or causes. CANDIES, which currently has 100 nodes and 400 causal links, deals with only 17 symptoms and already has performance problems running on a VAX (Longoni and

others, 1987:297). This ambitious project to totally automate a satellite may take more processing power than is currently available.

Neural Networks

Neural networks have a structure similar to that of causal networks. The weights assigned to the arcs on a causal network are similar to the weights between nodes in a neural network. In a causal net the weights on the arcs are assigned by a human expert, and each node output value is determined by hypothesis testing. In a neural net, the weights are determined by "training" the network and the output-value of each node (or neuron) is determined by a non-linear function of the inputs.

There are many different types of neural nets. One of the most popular is the back-propagating net. This net is trained by putting a known pattern onto the input layer (these could be a set of symptoms) and the desired output pattern onto the output layer (which could be a cause of the symptoms). After a random value is assigned to the weight of each arc, the weights on all the arcs in the network are adjusted by propagating values from the output layer back through the hidden layers to the input layer. Typically, this is done thousands of times until the weights reach a stable state and the desired output can be produced from that particular input pattern to a high degree of certainty. Once trained, the weights remain fixed and when that input pattern occurs again in a fault scenario, the output should respond with the correct diagnosis. An advantages of neural nets is that the input does not have to be exactly the same as the one used for training. Therefore, the net can deal with a degree of uncertainty.

A trained neural net is a fixed entity and cannot normally be changed on the fly. Neural nets can be designed in many different ways and can therefore be customized for different applications. Their many different topologies, training methods and activation functions allow architectures that look nothing like causal

networks. For example, another method for training networks is to have the nodes in the input layer represent time points. This allows transient data to be detected; such data is often a first indication of trouble. Dietz used this temporal technique as one of his neural net training methods for fault diagnosis on a jet engine (Dietz, 1988:18).

Seifert demonstrated the feasibility of using neural nets in the detection of satellite anomalies in his master's thesis. He applied neural nets to a particular anomaly in the Inertial Measurement Unit of the Attitude Determination and Control Subsystem that is not detectable by the flight software (Seifert, 1989).

Although neural nets can be trained to handle fault patterns that are similar to those used during training, they cannot handle fault patterns that are too different and not accounted for in the training process. Like rule-based expert systems, neural nets do not understand the underlying function and structure of the system. Like causal networks, neural nets could be trained to account for most anomalies that may occur, but they soon become very large and the extra training time required becomes impractical.

Model-Based Expert Systems

Rule-based expert systems, causal networks and neural nets have knowledge about the way systems fail. That is their main downfall; the number of ways a system can fail, and all their associated symptoms, can quickly grow to unmanageable levels. A model-based expert system, on the other hand, has knowledge about the way the system works. Model-based reasoning is a deep reasoning method that uses knowledge about the system's structure and function to diagnose faults. A simulation model of the system is operated alongside the real system. Continuous comparison of sensors in the real system to those in the model provides an alarm when an

anomaly occurs. The discrepancy between the real and model sensors initiates a diagnostic search (the model-based reasoning) for the culprit components.

Another major problem with rule-based expert systems is that they depend very much on the integrity of the sensors (Scarl and others, 1987:360). To cope with this problem, process monitoring expert systems typically have up to three-quarters of their rules written for sensor validation (Fulton, 1990:49). Since a sensor is just another component in a model-based expert system, validation occurs automatically by the reasoning mechanism. In a sensor-rich environment, model-based reasoning can detect non-intuitive anomalies and even failures that were not considered during the designer of the system (Fulton, 1990:55).

Some authors have stated that model-based expert systems are more complicated, difficult to set up, more costly, more inefficient for rapidly changing inputs and more limited by onboard computing power than rule-based expert systems (Rampino, 1987:19; Passani and Brindle, 1986:255; Longoni and others, 1987:295). The knowledge for the design of the model comes from the engineering schematics used to design the system. Therefore, an expert is not required to help with the design of the knowledge base. Knowledge for rule-based expert systems comes from an expert who has much experience in the domain of interest. Although it is easier writing rules than designing a computer model, the rules come from a knowledge acquisition phase that is usually a very difficult interaction between a knowledge engineer and the domain expert. The success of this interaction is crucial to the quality of the final product.

Model-based reasoning does have its drawbacks; if there are no (or very few) sensors in the system, or if there is no way of probing into the system, or if a complete and accurate model cannot be built, model-based reasoning cannot work successfully (Fulton, 1990:55). The lack of sensors is a major drawback for any diagnostic system (even a human), and is not peculiar to model-based systems.

Following are examp'es of two important expert systems that apply model-based reasoning to the detection of faults on space systems:

LES. The LOX (liquid oxygen) Expert System is a process monitoring and fault location system that has been developed at the Kennedy Space Center to monitor the loading of liquid oxygen onto the space shuttle. The domain includes analog and discrete inputs (commands) and sensors, and other objects such as transducers, relays, solenoids, valves, etc. These objects are represented in the model using a Frame Representation Language (FRL). The user interfaces with the system using on-screen schematics, which are generated automatically from a database, in block diagram form if desired. Discrepancies between sensors and the model are highlighted, and when selected with a mouse, the object can be tested. LES runs on a dedicated Symbolics 3600 and diagnosis takes from 10 to 45 seconds (summer 85) (Scarl et al, 1985:416).

PARAGON. Ford Aerospace has developed a system, named PARAGON, that interacts with an expert system builder to model a satellite subsystem for automated fault detection and correction. PARAGON also generates LISP code to simulate the satellite subsystem to allow verification of the model. Once verified, the model becomes the knowledge base for an expert system in which modules detect and diagnose malfunctions and suggest corrective actions. Blasdel's paper (Blasdel, 1987) describes how PARAGON is used to model a satellite's electrical power subsystem.

Hybrid Systems

Rarely can a single AI technique produce the desired results. Many researchers have realized this and have combined several AI techniques to achieve their aims. The diagnostic systems below are examples of hybrid systems that makes use of the rule-based and model-based techniques discussed above:

ISA. The Integrated Status Assessment expert system prototype was built to perform Station-wide failure diagnosis on the Space Station. It is a small part of MITRE Corporation's development of the Operations Management System (OMS) which automates many aspects of flight control for the Space Station on-board systems (Marsh, 1988:60). The prototype's domain is the communications and tracking system. The ISA system is designed as a hybrid expert system that uses both rule-based programming and qualitative modelling. Qualitative modelling is a model-based reasoning approach that uses terms such as "good" or "bad" to describe components instead of numerical information that would be used in a quantitative model. Interestingly, ISA uses deep reasoning to reduce the amount of rules for its rule-based expert system and to "home-in" on a specific component. The rule-base (shallow reasoning) is then used to reason *about* that component. This is the reverse of most hybrid systems that use deep and shallow reasoning. Marsh claims: "The shallow reasoning rules can handle many situations that evade deep reasoning rules which depend upon specifics about a component's structure" (Marsh, 1988:70). The prototype was hosted on a symbolics 3600 series computer and written in ZetaLisp and OPS5.

ACES. ACES is an Attitude Control Expert System prototype that detects problems in the momentum wheels of the DSCS-III satellite. This system uses telemetry tapes for its input and reasons about the data received using a rule-based expert system and quantitative and qualitative modelling techniques. In this case mathematical models are used to describe the function of the momentum wheels. The rule-base is used to hypothesize potential faults and the device models are used to confirm or deny them. The system contains approximately 50 rules and can correctly identify any problem with the reaction wheels which an attitude control simulator can simulate. (Passani and Brindle, 1986)

Seifert's thesis, discussed above under neural nets, describes a hybrid system. Following detection of a malfunction by the satellite's flight control software, Seifert's rule-based expert system determines what tests must be performed by the satellite software. The expert system and neural net then analyze the results to determine the corrective action (Seifert, 1989:3).

Summary

This chapter has looked at some current work using AI techniques in the diagnosis of satellite anomalies. All the examples given are listed as prototypes, so it appears very few are actually working in an operational environment. The techniques discussed have advantages and disadvantages, hence the need for hybrid systems. From the literature searched so far, it seems that the technique best suited for anomaly detection on an autonomous satellite is model-based reasoning. Model-based reasoning uses knowledge about the way a system works. The amount of knowledge required for this type of system is small when compared to the alternatives that require more knowledge about how a system can fail. Sensors compound the problem. They must be considered as components that can fail. For rule-based systems, or other systems that rely on knowledge on how a system can fail, sensor validation can result in a diagnostic system that can quickly become unmanageable. Model-based reasoning can only work successfully in a sensor-rich environment with complete and accurate models of components in the system. To achieve total autonomy, a satellite must be designed from the ground up to include the sensors, redundant components and the diagnostic engine to detect and recover from anomalies. Of course, this will add extra expense to the spacecraft, but this must be traded against the benefits to determine its worth.

III. Model-Based Reasoning

Introduction

There are a number of ways that model-based reasoning can be accomplished. This chapter reviews methods based on work done in the early 1980s using a constraint propagation technique. The technique is not without its problems, but over the last several years, on-going research has overcome some of those early problems and makes the technique suitable for the detection of anomalies on satellites. The purpose of the research for this chapter was not only to look at the various approaches to model-based reasoning but also to help determine the best approach to use for the detection of satellite anomalies.

The chapter begins with a general discussion of model-based reasoning and some special problems pertaining to satellite anomaly detection. This will be followed by a description of the basic technique pioneered by Davis and others, a discussion of its strengths and weaknesses, and a look at some of the variations of the technique that make it attractive for the domain of satellite anomaly detection.

Model-Based Reasoning

Model-based reasoning is also known by other names: "deep reasoning" and "reasoning from first principles" are popular in the literature. The term *deep reasoning* is intended to separate it from *shallow reasoning*. Shallow reasoning usually refers to empirical or rule-based expert systems that have no real understanding of system they are trying to diagnose. *Reasoning from first principles* is another phrase used to describe model-based reasoning, and tries to convey the idea that the reasoning mechanism has a fundamental understanding of the device in question.

Depending on the problem domain, the description of the device can be modelled using mathematical functions (such as differential equations) or using propositional logic.

The knowledge representation used to describe the model, and the reasoning mechanism used to determine which component is faulty, usually depend on the problem domain. However, much of the current research effort in model-based reasoning is concentrating on coming up with a domain independent inference engine for fault diagnosis. No matter how the knowledge is represented, or how the inference engine operates, the essence of model-based reasoning is that a model of a real-world system is constructed in a computer; this model simulates the real-world system's structure and behavior. The simulation is used to predict the way the real-world system should operate. If there is a discrepancy between the operation of the real-world system and the computer model then the diagnostic reasoning mechanism attempts to determine which component, or components, is causing the discrepancy.

In the early years, most research on model-based reasoning was applied to the fault diagnosis of electronic circuits (mainly digital systems). Many subsystems on satellites include control systems and other closed loop systems that are analog in nature. The number of sensors, and their placement on the satellite, are fixed. Obtaining data from places where there are no sensors is not possible. In contrast, technicians troubleshooting electronic circuits can be guided by a model-reasoning system to probe anywhere in the circuit.

A model, by definition, is only a representation of something else, and no model can represent the real world system exactly. Fortunately, the model need only be as accurate as required to allow for the detection of faults that would cause a shift in some intended or specified operation of the system. In an autonomous satellite, fault detection would only be required for a component or subassembly that can be replaced or reconfigured automatically onboard. More sensors would allow detection

of faults at a higher resolution, but fault diagnosis of these lower level components is simply not necessary. In space, nothing could be done about it.

Diagnosis of faults on a satellite is a low-resolution problem. However, the same reasoning techniques used to determine faulty components in electronic circuits could be used for the larger subassemblies on the satellite. For example, a servo-amplifier used on a satellite's attitude control subsystem would be treated similarly to an AND or OR gate in a digital logic circuit. The fact that one is analog and the other digital in nature just means that there will be differences in the functional description of the components, but not necessarily in the reasoning process.

Randall Davis(Davis, 1984)

In 1984 Randall Davis, an AI researcher at MIT, produced a paper called "Diagnostic Reasoning Based on Structure and Behavior" which spawned considerable follow-on research efforts in this area, judging from the many authors that have cited his work. Davis built on earlier research done by AI pioneers such as J. De Kleer (de Kleer, 1976; de Kleer, 1979; de Kleer and Brown, 1982), G. Sussman (Sussman and Steele, 1980), J.S. Brown (Brown, 1982) and M. Genesereth (Genesereth, 1981). His paper focused on the diagnosis of faults in digital logic circuits and uses the technique of *constraint propagation* and *constraint suspension* to generate a list of candidate components.

The process of *candidate generation* (determining which components may be failing) and *symptom generation* (how they may be failing) is what separates Davis's technique for troubleshooting from the traditional techniques which use test generation and verification methods alone. Test generation and verification (setting known inputs to a device and observing the outputs) is still used in Davis's technique, but it is applied only after generation of the candidate list.

Structure

Davis describes the structure of his circuits in two ways: *functional organization* and *physical organization*. These produce two distinct interconnection descriptions. One represents knowledge at the schematic level and the other is a hierarchical description at the block diagram level. His components are built on three concepts: *modules*, *ports* and *terminals*. The modules are the components represented as a black box. The ports are where information flows into and out the module, and the terminals represent the physical connection which can be probed for measurement or superimposed with other terminals to connect modules. Any module may have substructure, that is, sub-modules contained within modules. Davis limits this down to the gate level. In his paper, Davis used the examples in Figure 1 to describe the terminology and hierarchy for his structure. The example LISP code in Listing 1 shows how the structure is built, and how the sub-modules are generated.

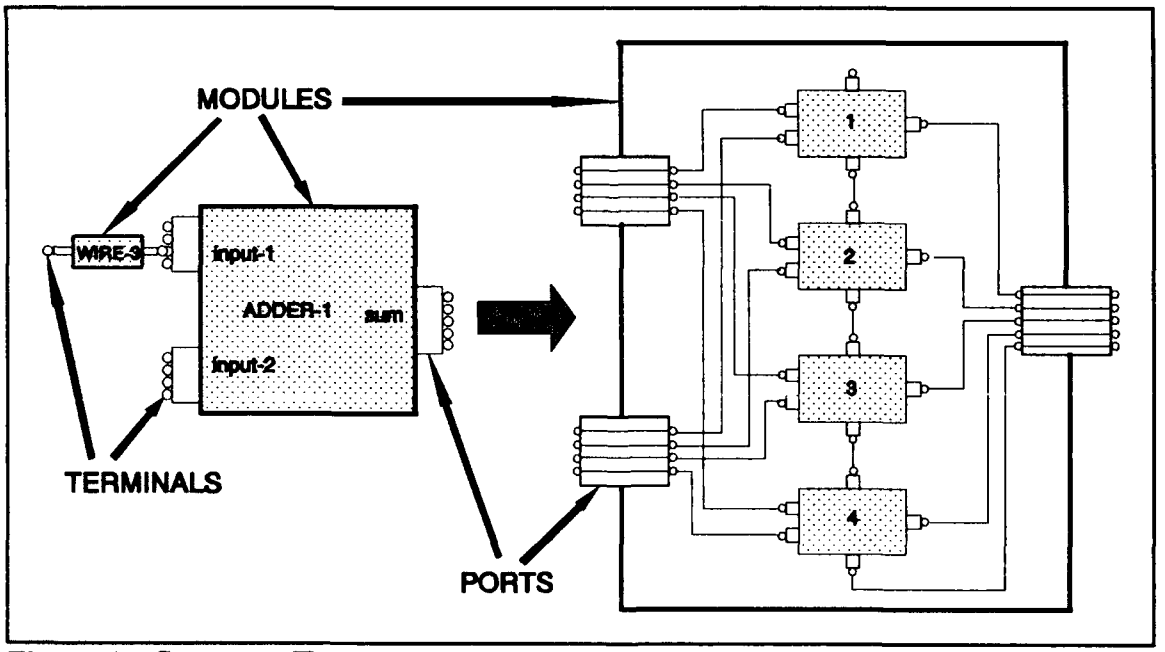

Figure 1 - Structure Terminology and Hierarchy (Davis, 1984:353,254)

```
(definemodule adder NBitsWide
   (repeat NBitsWide i
      (part slice-i adder-slice)
      (run-wire (input-1 adder) (input-1 slice-i))
      (run-wire (input-2 adder) (input-2 slice-i))
      (run-wire (output slice-i) (sum adder)) )
   (repeat (- NBitsWide 1) i
      (run-wire (carry-out slice-i) (carry-in slice-[i+1])) )
   (run-wire (carry-out slice-[NBitsWide-1]) (sum adder)) )
```

Listing 1 - Example LISP code that describes structure. (Davis, 1984:354)

Behavior

Following interconnection of its components, a device is modelled as a constraint network where each component's behavior is described bidirectionally. In the forward direction (the normal operating mode), the component's output is described as a function of its inputs. In the reverse direction, any input can be described as a function of its output and other inputs. For example, the normal (forward) operation of an adder could be described as:

$$Output = Input1 + Input2$$

In the reverse direction, the value of an input can be inferred from the output and the other input as follows:

$$Input1 = Output - Input2$$
$$Input2 = Output - Input1$$

In a constraint network, structure of the device is described by the connectivity between the individual components. The output wire from one component is connected to one or more inputs of other components. The behavior of each component is described as discussed above.

To simulate the real-world device, the inputs are set and allowed to propagate through all components to the output. These output values are compared to those measured on the physical device, and if any are not the same, a problem has been

detected and diagnosis is carried out to find the offending component. This is done by taking the measured output values and injecting them into the outputs of the model. These constraints are propagated backward through the model, and by switching out selected components in turn (*suspending* the component), and comparing the consistency-state of the model, a candidate list is generated. Components selected for suspension are all those upstream from where the discrepancy was detected. If several discrepancies are detected at the output, only the *intersection* of upstream sets need be suspended. This can save considerable computer time in complex systems.

The example circuit shown in Figure 2 was used by Davis (and several other authors since) to describe his technique for candidate generation. His procedure is detailed in Listing 2.

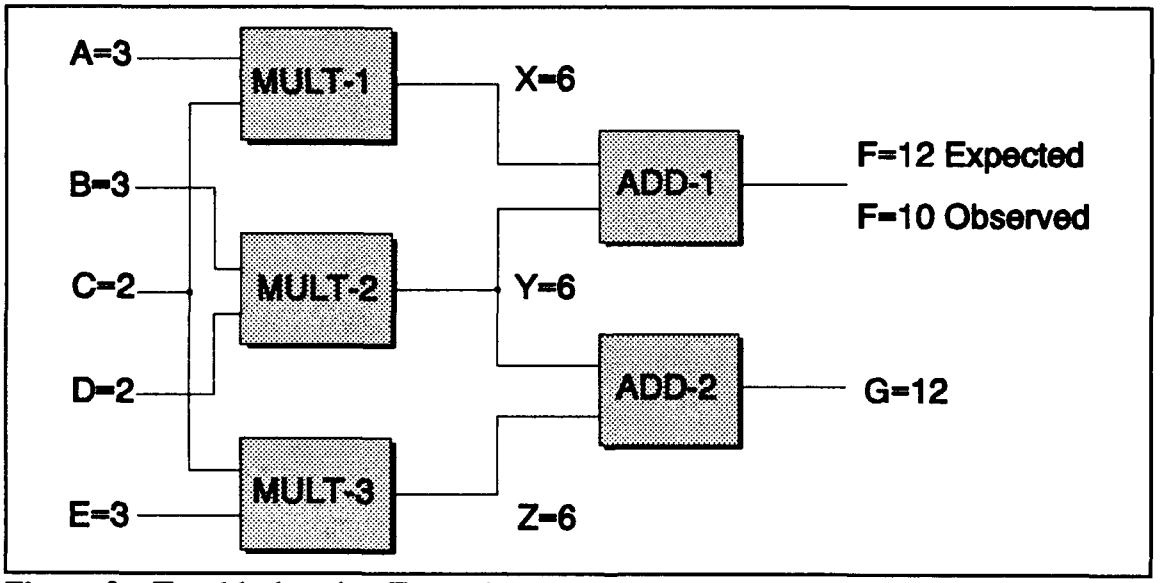

Figure 2 - Troubleshooting Example. (Davis, 1984:362)

STEP 1: Collect Discrepancies
1.1 Insert device inputs into the constraint network inputs
; e.g. insert 3, 3, 2, 2, and 3 at primary inputs A through E
; simulation predicts values at F and G
1.2 Compare predicted outputs with observed and collect discrepancies
; e.g., prediction and observation differ at F.

STEP 2: Determine Potential Candidates via Discrepancy Records
2.1 For each discrepancy found in step 1:
follow the dependency chain back from the predicted value
to find all components that contributed to that prediction
; these are all the components "upstream " of the discrepancy
; e.g., if we follow the dependency chain back from the 12
; at F we find adder-1, mult-1, and mult-2
2.2 Take the intersection of all the sets found by Step 2.1
; this yields the components common to all discrepancies (and
; hence potentially able to account for all discrepancies)
; (in the example above there is only one discrepancy)

STEP 3: Determine Candidate Consistency via Constraint Suspension
3.1 For each component found in Step 2.2:
3.1.1 Turn off (suspend) the constraint modeling its behavior
3.1.2 Insert observed values at outputs of constraint network
; (inputs were inserted earlier at Step 1.1)
3.1.3 If the network reaches a consistent state
-The component is a globally consistent candidate
-its symptoms can be found at its ports
-add the candidate and its symptoms to the candidate list
; e.g., adder-1 and its values of 6, 6, and 10
otherwise
the candidate is not globally consistent, ignore it
; e.g., mult-2
3.1.4 Retract the values at constraint network outputs
3.1.5 Turn on the constraint turned off in step 3.1.1
; (these last two just get ready for the next
; iteration of 3.1)

Listing 2 - Candidate generation via constraint suspension. (Davis, 1984:365)

Hypothesis Generation

Given a list of candidates, it is now time to generate hypotheses for component failures; these hypotheses will help eliminate most candidates. This is a difficult task because of the conflict between the desire to be complete, and the need for constraining the possibilities. Many assumptions are made for Davis's candidate

generation algorithm. A single point of failure, and non-intermittency are assumptions necessary for proper operation. However, other assumptions are not so obvious. Assuming the schematic is correct is not always a good assumption to make because there are paths of interaction between terminals that are not on the schematic.

For example, if power is not supplied to a chip (because of a broken or bent pin), an input wire can be pulled to ground resulting in an input terminal acting like an output. If this were to happen to Mult-3 then the lower input to Mult-1 would be zero. Another example is a bridge fault, where a solder splash shorts several pins or tracks on a circuit board. This could not be modelled using the structure and behavior methods already discussed, but can be taken into account with knowledge of how the chips are physically located. Wiring errors during assembly or even design errors are other problems that are usually assumed away because they don't exist on the circuit. As Davis puts it: "... the virtue of the technique is that it reasons from the schematic; the serious flaw in the technique is that it reasons from the schematic *and the schematic might be wrong*" (Davis, 1984).

In the examples just given, i.e., the input acting as an output and the bridging fault, it was assumed that no paths of causal interaction existed, yet pathways are a fact. Davis does not consider such assumptions to be a problem, and states that careful management of them are important to the reasoning process. Knowing how the modules interact with each other, e.g., close proximity, capacitive coupling, heat transfer, etc., it is possible to trace many paths of causality and come up with a wide variety of hypotheses. This could quickly get out of hand, resulting in every module's becoming a candidate, but if ignored some classes of faults may never detected.

Davis believes two steps can overcome this problem. First, it must be recognized that knowledge of the pathways of interaction is the source of the problem solving power and not the inference method (discrepancy detection and

constraint suspension). Second, the pathways of interaction must be somehow enumerated and organized so they can be used in the reasoning process.

How can interaction pathways be enumerated and organized for the reasoning process? Davis found the answer from methods used by human troubleshooters. Engineers know that some things go wrong more than others, so they employ simpler hypotheses first. The hypotheses are ordered, starting with the simplest first, but none is permanently excluded. Davis did this by enumerating the assumptions built into the model and built a list of failure-categories by violating each assumption. The order is fixed by the frequency of occurrence as reported by experienced troubleshooters. The list is built only by experience and uses no causality knowledge. For example, the order for the digital electronic domain is as follows:

— localized failure of function (e.g., stuck-at, failure of RAM cell)
— bridges
— unexpected direction (e.g., power failure problem)
— multiple point of failure
— intermittent error
— assembly error
— design error

The reasoning method uses the first assumption when generating the candidate list. If this leads to a contradiction, then this assumption is relaxed and the next assumption is used, i.e., that the problem may be a bridge fault. Davis believes that this is what a good engineer would do: "..make all the assumptions necessary to simplify a problem and make it tractable, but be prepared to discover that some of those simplifications were incorrect" (Davis, 1984).

Limitations

In his paper, Davis listed a number of problems and limitations with the technique. His examples used simple circuits to demonstrate the reasoning

mechanism. Davis admitted problems when scaling to real systems, such as a disk controller. He stated that the size of the system was not the main problem but that modelling the complex behavior would present a more difficult problem. Difficulty would arise deriving the inference rules used to infer the input of a component given its outputs. How do you infer the inputs of a disk controller given its outputs. Many devices are not invertible!

Another significant problem lies with modelling devices dependant on time and have state. Modelling propagation delays would be necessary to detect race conditions in digital circuits. Reasoning about protocols in communication systems are also important. Another problem is reasoning about devices that have memory. To overcome this, Davis hinted that future work could reason by moving back through time, similar to his movement back through space (the circuit) from the original discrepancy, to generate a candidate list.

Davis's work concentrated on digital circuits and he discusses additional problems associated with the modelling of analog circuits. Most analog devices are bi-directional. This makes candidate generation less constrained because there are more paths in the circuit resulting in many more components on the candidate list. Another problem with analog circuits is that they often use feedback or hysteresis and depend on a number of components to achieve a desired transfer function. A fault with the feedback system is difficult to trace to a particular component because that component will cause the whole feedback system to fail. Analog circuits also work with continuous values rather than the simpler on/off values of digital circuits. Qualitative physics has been used to try and overcome problems associated with continuous values by giving them qualitative values such as high, ok, low, rising, falling, etc.

Probing of the circuit was not used in Davis's work, but he did state that it is an expensive operation compared to inference which is "free". He stated that it was

expensive to put cards on extenders and that probing could disturb the circuit. For the onboard satellite diagnoses problem, probing is not possible. Sensors are fixed and so probing and its associated expense is not an issue.

Although Davis stated early in his paper that fault models were not required in his method of troubleshooting, his categories of failure appear to be a similar list. The categories of failure were needed to reduce the paths of interaction to be considered, obviating the need for an exhaustive search. But Davis claims that progress was made in two areas. First, by defining a fault as any discrepancy, he could deal with a wider range of faults, and second, he provided a formal way of generating the candidate list. The fault models used in rule-based systems are derived in an ad-hoc informal way that come from observations in practice. Davis's failure categories are derived from assumptions underlying his representation, and provide a relatively systematic basis for enumerating categories of failure to consider.

This brief review of Davis's work was intended to provide a background into model-based reasoning and its problems for the review that follows

Scarl, Jamieson and Delaune(Scarl et al, 1985; Scarl et al, 1987)

Before Scarl's paper, most work using model-based reasoning for fault detection was applied to electronic circuits. Scarl was one of the first to use it for process monitoring and control applications. The detection of faults in a process monitoring and control system is comparable to the detection (and correction) of faults in an autonomous satellite system. Just as in a satellite system, a process monitoring and control system has numerous sensors that report the health status of the system at any given time. Both systems also have a set of commands that can change the configuration of the system. If an anomaly is detected, a configuration change can switch around a faulty component or switch-in a redundant one.

Switching around a faulty component usually means living with a slightly degraded system. Switching in a redundant component usually results in identical performance, but at the expense of extra hardware; weight is an important aspect of satellite design.

Scarl's work places emphasis on sensor validation. This was not the case with Davis's work on electronic circuits. His sensors are test equipment (voltmeters, ammeters, oscilloscopes, etc.) external to the device under test, and assumed to be correct. The importance of sensor validation cannot be understated. As was discussed in the last chapter, sensor validation was the nemesis of rule-based systems. Just as in process control applications, sensor validation plays a critical role in satellite fault detection.

Scarl's work (LES) applies model-based reasoning to the oxygen loading system for the space shuttle. A brief overview of this system was given on page 18 in Chapter II. In this chapter the method he uses will be discussed. Specifically, the knowledge representation he uses for the structure and function, and the algorithm used for his diagnosis technique will be examined.

Structure and Function

The model of the oxygen loading system, represents all replaceable electro-mechanical components and the relationships between them. Each object is modelled using a version of the Frame Representation Language (FRL), which is similar to an object oriented language. The frame includes slots (place-holders for variables or methods) that describes the object's structure and function. Scarl uses SOURCE, SOURCE-PATH, and STATUS to express how a component is controlled. SOURCE points to the source of energy (e.g., a power bus or pressure line). SOURCE-PATH is an expression with a boolean result that determines whether or not the component is currently connected to its SOURCE. STATUS quantifies the

object's state whenever SOURCE-PATH is on. Some example frame definitions are shown in Listing 3.

```
(DEFRAME pot
  (STATUS (CSTATUS valve)) ...)

(DEFRAME valve
  (SOURCE-PATH (AND (NOT (CSTATUS override))
                    (> (CSTATUS analog-command)
                    0 )))
  ...)

(DEFRAME closed-limit-switch
  (SOURCE-PATH (COND ((< (CSTATUS valve) 4) T)
                     ((> (CSTATUS valve) 10) NIL)
                     (T'?)))
  ...)))
```

Listing 3 - Example Frame Definitions (Scarl, 1987:362)

The first frame, representing a potentiometer, simply names the value. The pot directly reflects a valves position and the CSTATUS is a macro returning the valves expectation value. The second frame is for the valve, and its SOURCE-PATH expression is true (valve open) if an override closed switch is OFF and the analog-command is non-zero. The valve's STATUS is simply the analog command. The third frame example is for the closed-limit-switch, which is a discrete device. The *COND* expression says that this switch is expected to be ON whenever the valve is less than four-percent open, OFF when the valve is more that ten-percent open, and otherwise indeterminate.

It is a structural convention in Scarl's knowledge base that sensors control no other objects. If this proves difficult, virtual sensors are created. For example, ammeters or flowmeters are split into resistors or pipes with virtual attached sensors that would be diagnosed as separate functional components.

Scarl defines his objects as the smallest replaceable units. This means he does not descend to wire and bolt level description of components. This absence of hierarchical structuring allows his system to ignore wires and connectors resulting in

a compact representational scheme that suppresses unneeded detail. This representation seems attractive for a satellite system.

Definitions

Scarl makes the following definitions:

- A *system* is the subject domain being represented.
- A *network* is a complex physical system of functional relationships.
- *Consistent* measurements give information about a system that agrees with adjacent network components.
- *Discrepant* measurements give information about a system that does not agree with adjacent network components.
- *Commands* are inputs to the network.
- *Sensors* are outputs from the network.
- The *expectation value* is determined by propagating commands through the network. Expectation values are determined from information flow in the forward (normal) direction (command to object).
- A *hypothetical value* is determined by treating a measurement as a constraint and describing what some object's state is implied to be by observation. Hypothetical values are derived from a reverse direction (sensor to object).
- An *innocent* object in incapable of being the cause of a given sensor discrepancy.
- A *culprit* is demonstrably the cause of a given sensor discrepancy.
- A *suspect* may be the cause of a given sensor discrepancy.
- The *original discrepancy (OD)* is the first discrepancy to be noticed.
- *Siblings* of a given sensor are those other sensors that depend in any way upon any of the commands controlling that sensor.

Full Consistency Algorithm

Before examination of Scarl's Full Consistency Algorithm, his assumptions are listed as follows:

Assumptions.

- The domain's structure and its internal functional relationships are known. These may change since faults can be considered as changes in the system's structure or function, but the functional relationships must be known at any time within some specified tolerance.
- Only single points of failure will occur in an operational system. This says that the time between failures is long compared to diagnosis time.
- Sensor-based diagnosis is adequate. This says that behaviors lasting longer than the sensor polling cycle are handled, while shorter-lasting behaviors can be ignored. It also says that the system has sufficient sensors to make important discriminations so that the residual sets of suspects are acceptably small.

Limitations.

- The current implementation cannot handle feedback.
- Only single output objects can be used.
- Bridge or directional faults are not supported.

Scarl's algorithm is essentially a search for some fault that can explain what the sensors are showing. As can be seen in the algorithm shown in Listing 4, all components and all sensors are not tested. Only those objects structurally upstream from the OD need to be tested. But all those sensors downstream from the OD are not a sufficient set for correct diagnosis. All sensors that are in any way dependent on the suspects OD should be tested, i.e., the OD's sibling sensors.

Following detection of a sensor discrepancy:

1. Pick some object, structurally upstream from the OD and label it as a suspect.
2. Hypothesize some faulty state for the suspect. Since the suspect's model can say what its properly working state is, any other conceivable state could be chosen.
3. Given this hypothesis, the model is used to derive expectation values for the system's sensors.
4. These sensor's measurements are then tested for consistency with those expectation values.
    a. If all sensors are consistent with their expectation values, then the hypothesized fault is one possible explanation of sensed observations, and the suspect is retained.
    b. If any sensor reading is not consistent with its expectation value, then that hypothetical fault (just the fault, not the suspect) is ruled out.
5. If Case 4.b. applies, then continue from Step 2 with that same suspect and repeat for all possible fault hypotheses. If all its possible faults are ruled out, then the suspect is cleared.

Listing 4 - Scarl's Full Consistency Algorithm (Scarl, 1987:364)

This algorithm works equally well when a sensor is included in the suspect list. The suspect may the OD sensor itself. The structural convention that the sensor controls no other object implies that (a) there is always one and only one sensor among the suspects (the OD itself) and that (b) the sensor is cleared if any siblings has a discrepant reading. This makes sensor diagnosis very much more efficient than in other systems, such as forward chaining systems.

Diagnosis time depends on the number of suspects and siblings, which in turn depend on the depth and branching factor of the control lattice (not the size of the knowledge base).

More Recent Work in Model-based Reasoning

The rest of this chapter very briefly discusses more recent work in model-based reasoning and is included here only as a point of reference.

De Kleer and Williams(de Kleer and Williams, 1987)

De Kleer and Williams built on the work of Davis, but are able to diagnose *multiple* faults in a digital circuit. Constraint propagation is coupled with an Assumption-based Truth Maintenance System (ATMS) for managing different diagnostic hypotheses. Their paper describes the General Diagnostic Engine (GDE), and their research delves into:

- Minimal sets
- Exploiting the iterative nature of diagnosis.
- Separating diagnosis and behavior prediction using a domain independent diagnostic procedure.
- Combining of model-based prediction with sequential diagnosis to propose measurements for fault localization.
- Use of probabilities and information theory in their diagnosis.

The main problem with their system is that it generates many alternatives; it is combinatorially explosive! This is largely overcome by using fault models, discussed in a later paper by De Kleer and Williams.

Struss and Dressler(Struss and Dressler, 1989)

Struss and Dressler recognized that the techniques used by de Kleer and Williams often generated candidates that could not possibly have caused the fault. This is because de Kleer and Williams's General Diagnostic Engine (GDE) captures only the correct, or intended, behavior of its components and does not have any

knowledge about how components behave when they are faulty. Struss and Dressler enhance GDE to GDE+ by incorporating fault models to help rule out any implausible diagnostic hypotheses and prove the correctness of components.

The whole advantage to model-based reasoning however, is that knowledge about the way a component fails is not required. After all, wasn't that the downfall of rule-based expert systems and causal nets? That's true, but those methods of fault diagnosis did not use model-based reasoning in the first instance. Once the model-based reasoning mechanism provides the candidate list (which is large for GDE since it can handle multiple faults), knowledge about the faulty behavior of components is exploited to cut down the size of the candidate list.

In their paper, Struss and Dressler provide a very good example to demonstrate GDE's lack of knowledge of component faulty behavior. They use a battery and three light bulbs connected as shown in Figure 3. The battery, S, is connected

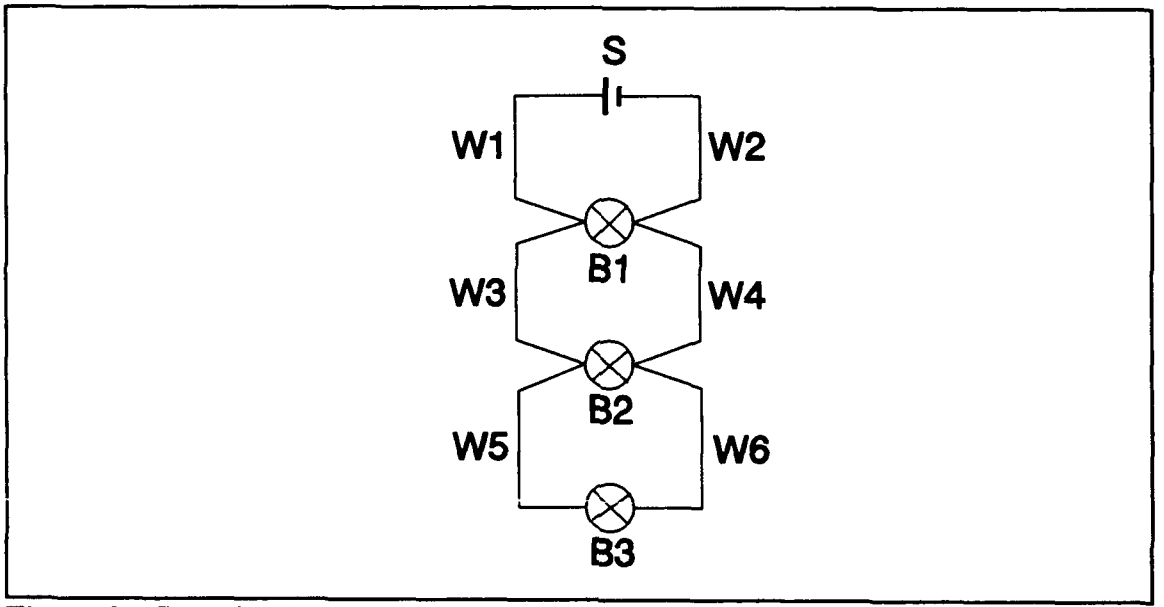

Figure 3 - Struss' light bulb example. (Struss and Dressler, 1989:1318)

to three bulbs, B1, B2, and B3, of the same type. Suppose it is observed that B3 is lit. Without requiring further measurements, a plausible diagnosis is that B1 and B2

are broken. This is because B3 indicates that battery is ok. GDE would suggest a variety of diagnoses. Among them is the preferred diagnosis of B1 and B2, but it will also show S and B3, W1 and W5 and many others. The diagnosis S and B3 explains the observations by a fault in the battery and the light bulb B3; "The battery does not supply voltage that is why B1 and B2 are not lit) and B3 is faulted: it is lit although there is no voltage"! This is of course is a ridiculous situation, and the explanation would not be accepted, because, unlike GDE, we use not only knowledge about the correct behavior of the involved components, but also knowledge about their possible behavior when they are faulted. For example, being lit without a voltage supply is not a potential fault of a light bulb.

This example demonstrates that knowledge about the faulty behavior can be very important. In many cases, the diagnostic process involves both

- generating candidates, or diagnostic hypotheses, by identifying sets of components whose correct functioning contradicts the observations, and
- generating explanations, or confirming diagnoses, by analyzing whether the malfunctioning of a (set of) components(s) is consistent with the observations.

Thomas Adams(Adams, 1986)

Adams's 1986 paper describes a practical application of model-based reasoning for automated fault diagnosis and load management for a spacecraft electrical power system. Adams describes a domain-independent inference engine that diagnoses faults using a constraint propagation mechanism. Similar to Davis's work, the diagnostic procedure consists of establishing consistency between the predictions of the constraint network and the observed data. The paper also discusses recovery planning and its effects on diagnostic activity.

A spacecraft power system consists mostly of analog components; the knowledge describing the behavior of the components and the interaction of interconnected components is expressed as constraint relationships. The components are modelled using current-voltage relationships (Ohms Law) and their interconnections using Kirchoff's nodal current law. Nonlinear models are treated similarly but use piecewise-linear models instead.

James Skinner (Skinner, 1988)

Skinner's thesis describes a blended diagnostic system which combines shallow (rules) and deep (models) reasoning to diagnose faults in a submarine inertial navigation system. Skinner's deep reasoning relies mainly on connectivity of components and not their function, and so is limited in its reasoning capability.

Raymond Yost (Yost, 1989)

Yost's thesis applies Davis's techniques to the same Inertial Navigation System used by Skinner, and thus overcomes some of Skinner's limitations. He used AI Squared Inc's Intelligent Diagnostic Expert Assistant (IDEA) tool to model the INS system because it was an implentation of Davis's work. But in his thesis, Yost stated the following limitations with this system:

Difficulty with Feedback Loops. Like Skinner, Yost had difficulty modelling circuits that had feedback loops. The version of IDEA he used did not help in this regard. He overcame the problem by modifying the representation of the circuit, but in doing so, he lost the structure of the real system.

Lack of Hierarchial Strategy. Again, Yost was constrained by the IDEA model-based reasoning tool. This tool did not allow different levels of abstraction - the way a technician works to break down a problem.

No Heuristic Reasoning Ability. Yost found numerous instances where simple heuristics could have pruned out the problem space before resorting to a more detailed diagnosis using a model. His model-based prototype, and the IDEA tool had no provision for heuristic reasoning ability. He recommended that any future model-based tool should include some rule-based capability.

Chen and Sargur (Chen, 1989)

Model-based fault diagnosis generates a list of candidates that are potential causes of the fault. Chen's paper provides a method for looking at the most likely candidate first in order to cut down on the computation required if every candidate were looked at exhaustively.

Gallanti (Gallanti, 1989)

Gallanti claims that application of model-based reasoning to real-world problems is still poor. His method make it practical for diagnostic problem solving in automated systems for monitoring continuous processes (Applicable to spacecraft). His system uses different levels of abstraction. Qualitative causal models (e.g., hi/low/ok etc) are used for candidate generation and quantitative models (real values) are used for validation or rejection of candidates.

Dvorak (Dvorak, 1989)

Dvorak's paper is on fault diagnosis of continuous-variable dynamic systems (CVDS). His method describes diagnoses of CVDSs where values are continuous (not discrete) and constantly changing, relatively few parameters are observable, and diagnosis is performed while the system operates. The method exploits the system's dynamic behavior using qualitative and quantitative models. Diagnostic knowledge is produced by qualitative simulation, continuously comparing observations against

fault-model predictions, and incrementally creating and testing multiple-fault hypotheses. The diagnosis is refined as the physical system's *dynamic* behavior is revealed over time.

Summary

This chapter has examined model-based reasoning techniques presented by many authors. Davis's and Scarl's work was discussed in some detail. Davis's work was examined because it provided a good basis for the understanding of the model-based reasoning paradigm. His work was applied to electronic circuits that could easily be probed for more data, so it really is not directly suitable for remote systems such as a satellite. Scarl's work was examined in detail because it was applied to a process control system. A satellite is similar to a process control system because it has many (fixed) sensors and further probing is not usually possible. Davis's and Scarl's methods have their limitations. In particular, limitation with feedback systems and systems that have state, or are otherwise time dependent. Many AI researchers are working in the area of model-based reasoning in an effort to overcome some of these limitations. A brief description of some of their work has also been presented. Appendix M lists a summary of research papers, in chronological order, of work done in the field of model-based reasoning.

IV. Satellite Subsystem

Introduction

The aim of this chapter is to present an overview of a major satellite subsystem and to give a more detailed look at a part of that subsystem that may used by a model-based reasoning fault detection system. The subsystem chosen was the Attitude and Velocity Control Subsystem (AVCS) of a typical geo-stationary communications satellite. Because of complexity and thesis-time constraints, only the pitch control channel of the AVCS is modelled and described in detail. However, an overview of the AVCS will be given to put the pitch control channel into perspective and to give the reader an understanding of how it fits in with the rest of the AVCS subsystem.

Information for this chapter was taken from an Orbital Operations Handbook (OOH) of a communications satellite. Although the orbital handbook provides a great quantity of detail, the information obtained is only used in a generic way. The handbook provided guidance on how such a subsystem operates and how it is implemented in a real spacecraft. Most importantly, the handbook provided realistic values for such things as pitch and roll error angles, control voltages, wheel speed, and their tolerances. These values were important in the model-building process to ensure the design of a realistic model.

General Description of Satellite

The spacecraft is a communications satellite operating in geo-stationary orbit. Like most communications satellites, this satellite is controlled from the ground by commands sent over a telemetry link. Satellite status is sent to ground controllers from onboard sensors via the telemetry link. If anomalies are detected, and the

problem proves to be a faulty module, controllers can switch to a redundant module within the subsystem to circumvent the problem. To ensure long-term (five years) reliability, the satellite has two of almost everything. The "A" module is the primary and the "B" module is the redundant backup. Primary or redundant modules are selected from decoded commands that activate relays to connect or remove power from the modules.

Attitude & Velocity Control Subsystem (AVCS)

AVCS General Description

Following successful insertion of the satellite into the correct position of the geo-stationary orbit, the primary function of the attitude and velocity control subsystem (AVCS) is to keep the satellite's antennas pointing at the earth's center (*nadir*), and the solar panels pointing at the sun. The AVCS carries out many functions, especially before orbit insertion. The AVCS provides for attitude error sensing, data storage, signal processing, mode switching, verification of commands, and control of actuation devices. It operates in a spin-stabilized mode during the transfer orbit; it provides for spin speed adjustment, despin, and sun/earth acquisition control; and it operates in a three axis on-orbit control mode. This thesis is only concerned with the on-orbit "normal mode" of operation of the AVCS. Accurate three-axis pointing is achieved using a small body-fixed momentum wheel biased at 3000 rpm, in conjunction with low level (0.1 pound) hydrazine thrusters and an earth sensor utilizing only roll and pitch attitude sensing. The AVCS also provides for ground-commanded orbital velocity changes while in a three-axis thruster control mode. The orbital velocity changes are required for "station-keeping", that is, keeping the satellite in the correct position, both in latitude and longitude, in the geo-stationary orbit. This is required because the satellite moves out of its correct

position due to perturbations such as non-constant gravity field, solar wind, meteorites, and on-board movement (solar arrays).

The overall block diagram of the AVCS is shown at Figure 4. The figure shows the eight different assemblies associated with the subsystem. On the left of the diagram are the sensor assemblies, consisting of the earth sensor, the sun sensor

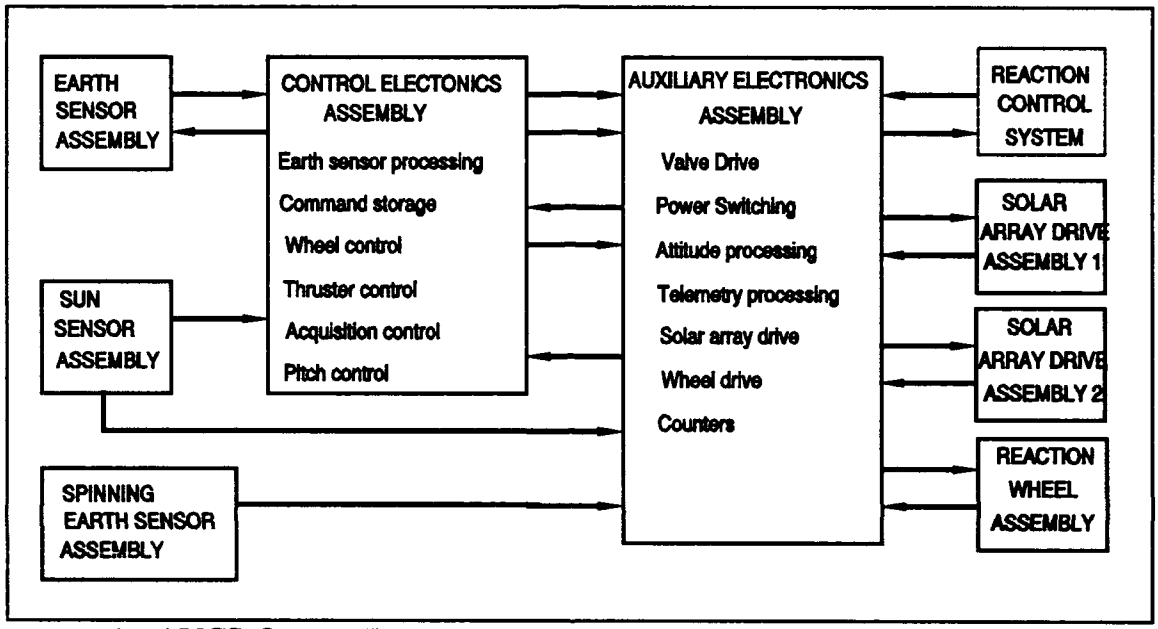

Figure 4 - AVCS Overall Block Diagram (Orbital Handbook: 3.2-5)

(including spinning and non-spinning detectors) and the spinning earth sensor (used for transfer orbit attitude determination). The two electronics assemblies, at the center of the diagram, process data from the sensors and carry out other functions as shown. The Control Electronics Assembly implements control laws through the various mission phases to cause the spacecraft to acquire the sun, to acquire the earth, to maintain earth-sun pointing, and to maintain pointing for normal on-orbit operations. The Auxiliary Electronics Assembly is the interface between the control electronics and other spacecraft systems; it primarily drives the actuators shown as the four boxes on the right of Figure 4. Note that the Reaction Control System (hydrazine thrusters), shown as a dashed box in the Figure, is not a part of the

AVCS; however, the drive electronics, that operate the thruster valves in the reaction control system, is part of the AVCS.

Coordinate System

A satellite's coordinate system is similar to that of an aircraft. Just as the belly of an aircraft points to the ground, the directional antenna of a communication satellite points to the ground. This line, to the center of the earth, is the positive Z-axis. Clockwise rotation around this axis, looking in the positive direction of the axis (towards the earth) is positive yaw, and is denoted as +Z or +Y. The satellite's direction of motion (the velocity vector) is the positive X-axis, and clockwise rotation

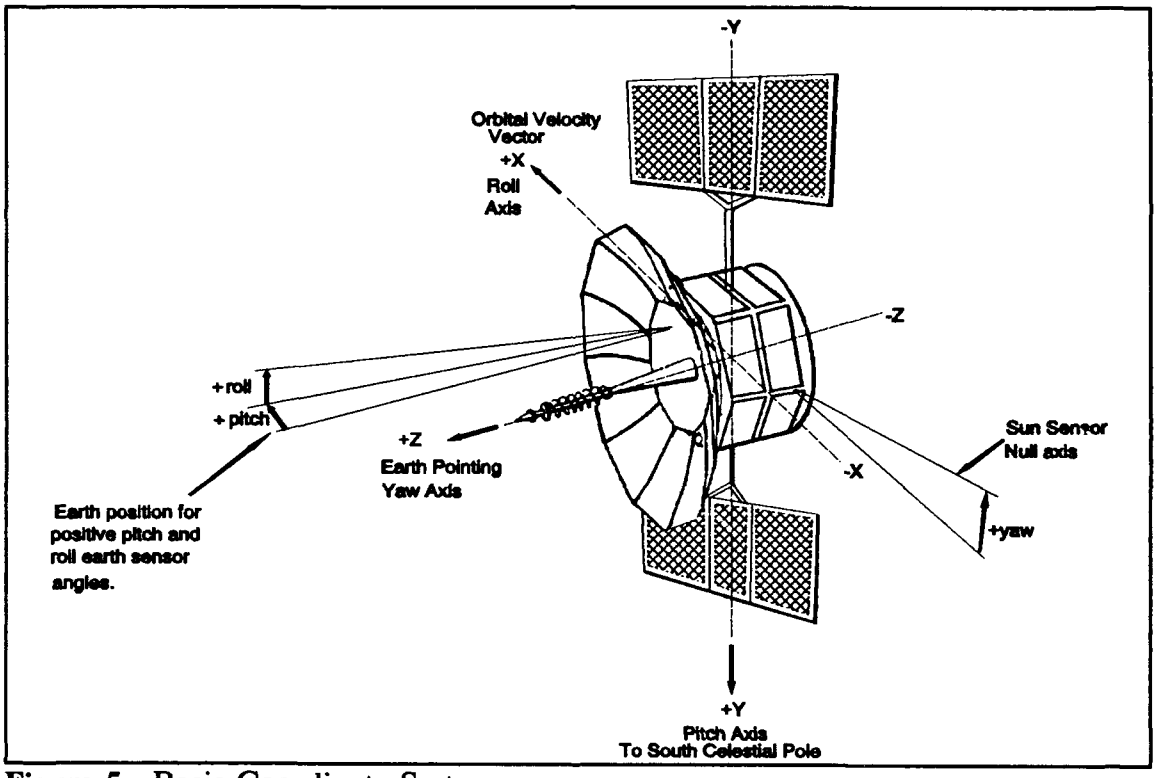

Figure 5 - Basic Coordinate System

around this axis is positive roll, denoted by +R. The axis perpendicular to the X-axis and Z-axis is the Y-axis. For a geo-stationary satellite, the Y-axis is parallel to the earth's rotation axis, and its positive direction points to the south celestial pole.

Clockwise rotation about the Y-axis is positive pitch, and is denoted as +P. The solar-panels are rotated about the Y-axis to ensure that the panels remain facing the sun during the satellite's 24 hour journey around the earth. The coordinate system is illustrated in Figure 5.

Reaction Control System

Attitude is controlled by hydrazine-fuel thrusters, strategically mounted (in pairs for redundancy) on the spacecraft to provide torques to rotate the craft about any desired axis. Placement of the thrusters on the spacecraft is illustrated in Figure 6. The satellite has two pairs of 1.0 pound thrusters for pitch (+P and -P), two pairs of 1.0 pound thrusters for roll (+R and -R), and four pairs of 1.0 pound thrusters for yaw (-Z1, +Z2, -Z3, & +Z4). To cause the satellite to spin (as is done in the transfer orbit) in the positive yaw direction, +Z2 and +Z4 would be fired. To stop spinning, the -Z1 and -Z3 would be fired. For velocity changes (as required for station-keeping), -Z1 and +Z2 would be fired to increase velocity in the X direction, and -Z3 and +Z4 would be fired to decrease velocity in the X direction.

The 1.0 pound thrusters are used for orbit insertion, for velocity changes, and for momentum wheel unloading. When the satellite is in normal on-orbit mode, these thrusters are disabled and 0.1 pound thrusters are used for roll and yaw stabilization (-RY and +RY). Pitch is controlled by a momentum wheel (of which there are two, for redundancy). The momentum vector points down the negative Y-axis, i.e., the wheel axis is in-line with the Y-axis and the wheel spins anti-clockwise when looking down the positive Y-axis (negative pitch rotation). Accelerating the wheel generates a positive pitch torque, conversely, decelerating the wheel generates a torque in the negative pitch direction. Figure 6 also shows the orientation of the momentum wheels on the spacecraft.

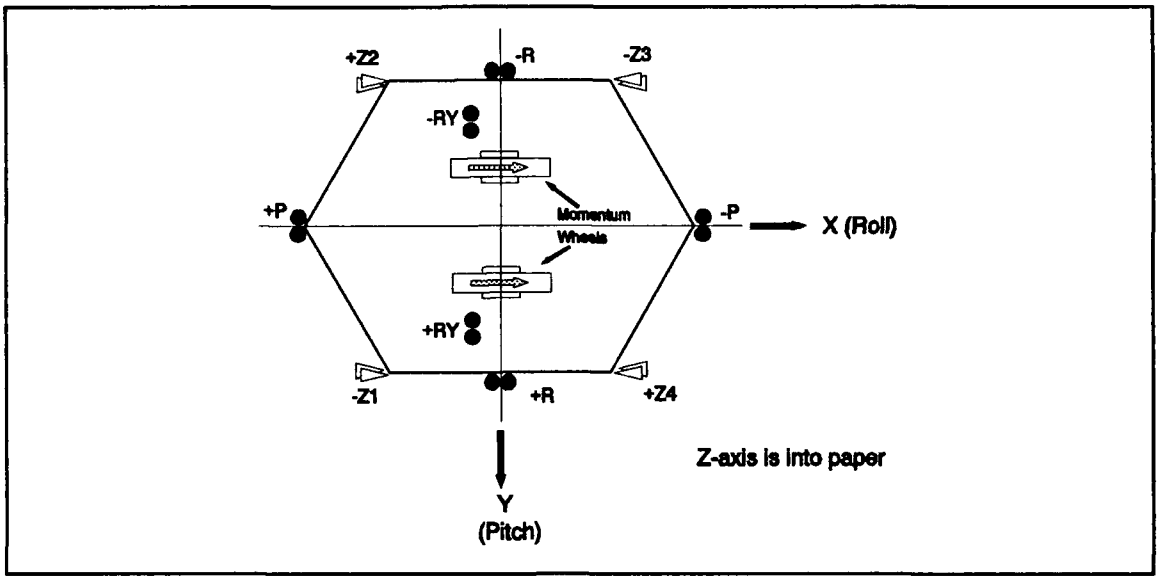

Figure 6 - RCS thruster and wheel locations (OOH:3.2-13)

AVCS Modes

The AVCS has to accomplish many operations before normal on-orbit mode is in effect. The satellite goes through seven distinct attitude modes from launch to its normal on-orbit operation. These modes, and the operations performed during each, are summarized in Table 1. Each mode represents a different hardware configuration that is manually selected by commands from the ground. An exception is the switch from mode 3B to 4A. This is an automatic mode change. This thesis is only concerned with mode 4B, the Normal mode.

Mode 4B - Normal Mode

The spacecraft operates in the Normal mode most of its mission life. The following functions are carried out during the Normal mode of operation:

- Normal control initialization (reaction wheel and 0.1 lb thrusters)
- Continuous monitoring of spacecraft state-of-health
- Payload configuration

Table 1
AVCS Modes (OOH 3.2-28)

Mode No.	Name	Operation
1	Launch/Spin-up	• Spacecraft spin-up about Z-axis
2A	Coast	• Spin speed adjust • Spin axis attitude determination • Spacecraft precession
2B	Despin	• Despin of spacecraft • Solar array reorientation
3A	Sun Acquisition	• Point X axis (plus or minus) at sun
3B	Earth Search/Acquisition	• Adjust roll rate • Search for earth and automatically switch to Mode 4A
4A	Sun/Earth Point	• 3-axis stabilize on earth using sun for yaw control • Perform orbit velocity change
4B	Normal	• 3-axis stabilize on earth using reaction wheel and roll/yaw thrusters

- Battery reconditioning
- Switching earth sensors (north and south)
- Adjustment of solar arrays
- Reaction wheel momentum unloading

To attain the design goal of a five year mission life-span, attitude pointing is accomplished with minimum propellant usage. Low propellant usage is realized by using a reaction wheel for pitch stabilization and by using low-level (0.1 pound) thrusters for roll (and some yaw) stabilization.

Pitch. The pitch error signal, coming from the earth sensor, controls the speed of a biassed (rotating at some nominal rpm) reaction wheel. An increase or

decrease in wheel speed depends on the polarity of the pitch error signal. Since the wheel axis is along the Y-axis, this results in a pitch torque on the spacecraft.

Roll and Yaw. The roll error signal drives the low-level roll thrusters which are tilted to provide a small component of yaw torque. These R/Y thrusters not only provide roll torque, but, in conjunction with the momentum vector of the biased reaction wheel, will provide damping so that the X-axis is maintained in the orbit plane as the spacecraft moves around the orbit. This obviates the requirement for continuous use of a yaw sensor.

A pictorial representation of the AVCS operation in the 4B-Normal mode is illustrated in Figure 7. AVCS hardware configuration for the pitch and roll/yaw channels for the 4B-Normal mode is shown in Figure 8.

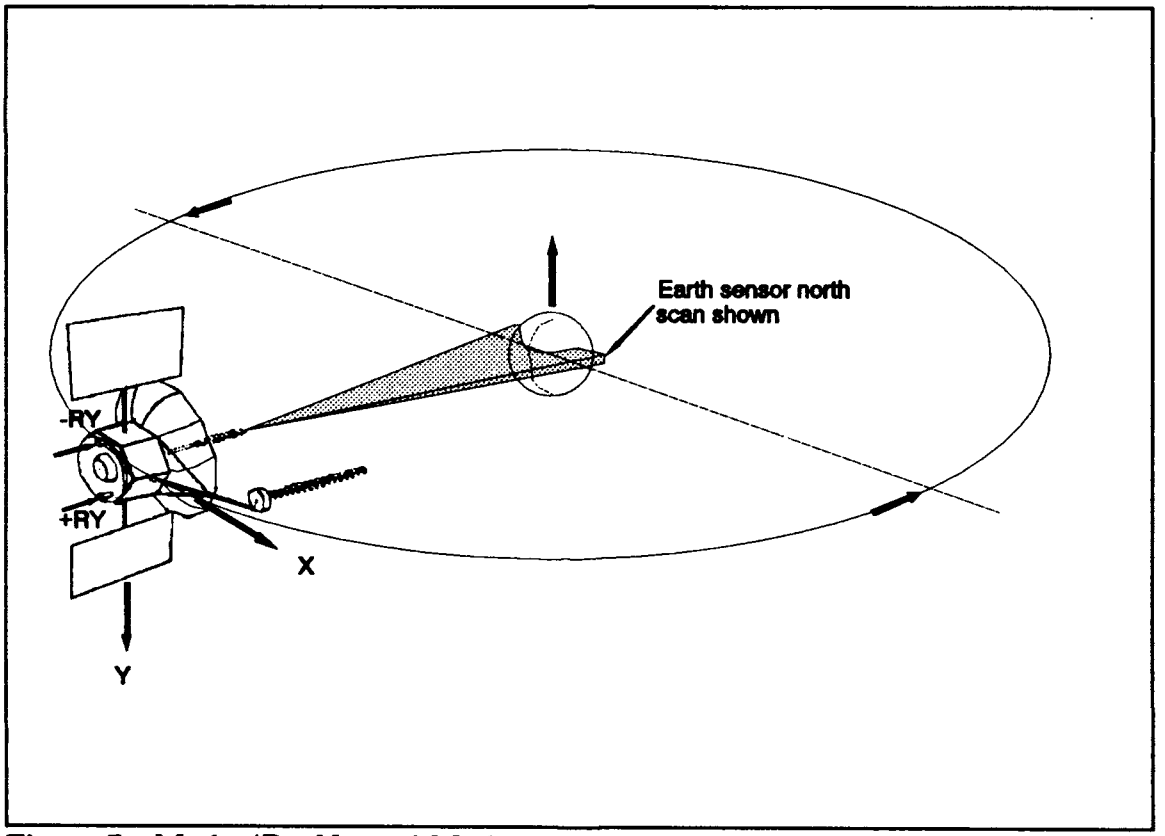

Figure 7 - Mode 4B - Normal Mode (OOH:3.2-35)

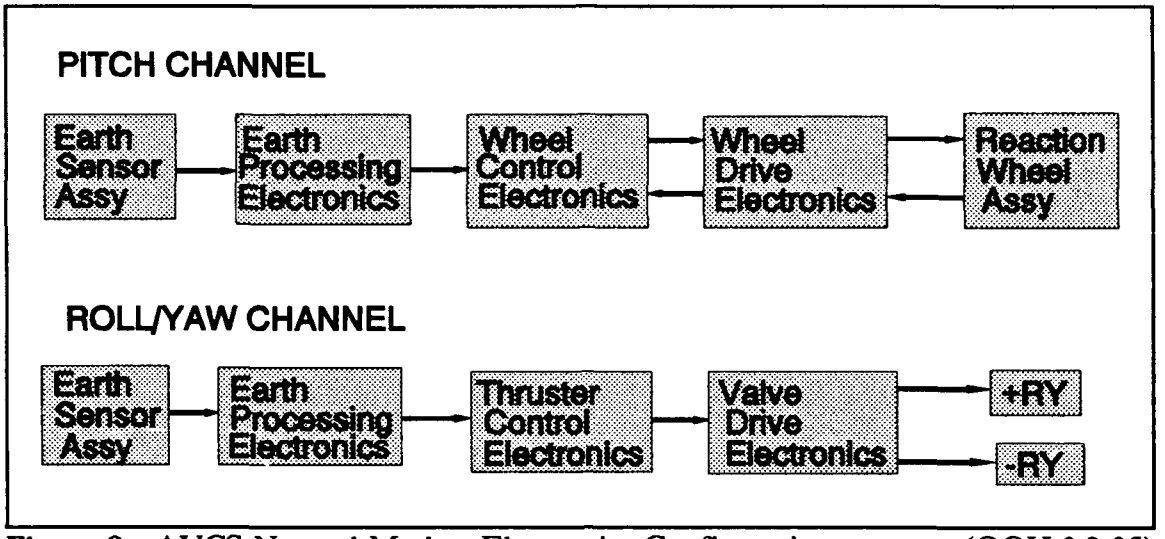

Figure 8 - AVCS Normal Mode - Electronics Configuration (OOH:3.2-35)

On-Orbit Pitch Control Channel

As Figure 8 illustrates, the pitch control channel is composed of five modules: the Earth Sensor Assembly (ESA), the Earth Processing Electronics (EPE), the Wheel Control Electronics (WCE), the Wheel Drive Electronics (WDE) and the Reaction Wheel Assembly (RWA). Since these will be modelled in the model-based reasoning software, the operation of each will be discussed in detail. However, before this is done, it is useful to look at the pitch control channel as a whole— its inputs, its outputs, and its function. A block diagram of the pitch control channel, showing cross strapping for redundant operation, inputs (commands) and outputs (sensors) is illustrated in Figure 9.

Inputs

Inputs to the pitch control channel come from two places. First, sensors provide the pitch error offset angle that is detected using the mirror scanning mechanics and electronics in the earth sensor assembly (ESA). Second, decoded ground commands select options or different configurations in the pitch control

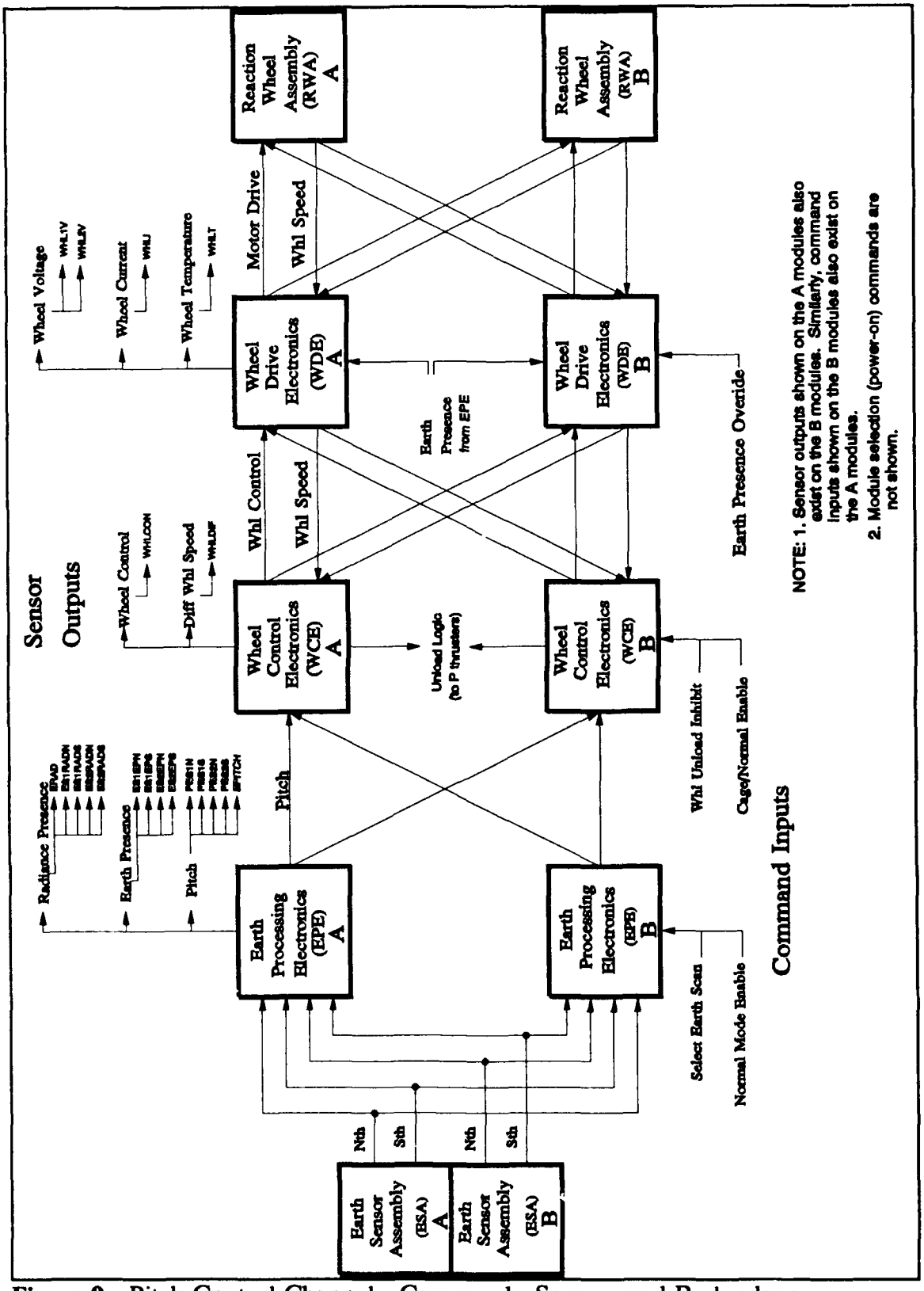

Figure 9 - Pitch Control Channel - Commands, Sensors and Redundancy

channel. These commands are important because after a faulty module is detected by the model-based reasoning software, these commands can be issued to correct for an anomalous situation. This is exactly what is done by ground personnel. A summary of all the commands that affect the pitch control channel is listed below:

- **Power Switching Commands.** These commands connect and disconnect power to primary and redundant modules when required.
- **Sensor Selection Commands.** These commands select one of four earth scanning beams in the earth sensor assembly. The A and the B earth sensor each produce a north and south beam, and both sensors are always active.
- **Normal Mode Selection Command.** This command puts the AVCS into the mode 4B configuration. The 1.0 pound thrusters valves are disabled and the 0.1 pound RY thrusters valves and reaction wheel are enabled. In addition, this command also inhibits WHEEL CAGE.
- **Normal Mode Disable Command.** This command disables the previous command function.
- **Wheel Unload Normal Command.** This command enables the wheel unload output in the wheel control electronics (WCE) so that pulses can be sent to the ±P valve drivers (1.0 pound pitch thrusters) to effect an unloading of the wheel momentum. This is periodically required to keep the wheel biassed at its nominal speed (3000 rpm).
- **Wheel Unload Inhibit Command.** This command inhibits the wheel unload normal command.
- **Earth Presence Override Command.** This command overrides the earth presence signal from the selected earth sensor scan so that presence is always indicated. This command drives the wheel drive logic as an earth presence signal.

- **Earth Presence Normal Command.** This command disables the previous command function.
- **Wheel Cage Command.** This command disconnects the wheel from the control loop and forces it to be driven to, and held at, its nominal rpm.
- **Wheel Normal Command.** This command puts the wheel back into the pitch control loop.

Outputs

Outputs from the pitch control channel also come in two forms. First, an important output is the torque that moves the spacecraft in pitch attitude. This is done by speeding up or slowing down the momentum wheel. Second there are many outputs that come from sensors throughout the pitch control channel. These outputs are telemetered to the ground, and monitored for spacecraft health status. These sensor outputs are important because they can be used by the model-based reasoning software to help determine which module is the culprit if an anomaly is detected in the pitch control channel. This is exactly what is done manually by ground control personnel. A summary of all the sensor outputs that affect the pitch control channel is listed below:

- **Radiance Present.** The following outputs indicate if earth radiance exists anywhere in the beam. Output is bilevel where 1 = Present, 0 = Not Present.
  - — **ERAD Radiance Present.** Indicates if earth radiance is detected in any of the Earth Sensor Assembly's (ESA) four scans.
  - — **ES1RADN Earth Radiance Present.** Indicates if earth radiance is present in ESA(A) north scan.
  - — **ES1RADS Earth Radiance Present.** Indicates if earth radiance is present in ESA(A) south scan.

— **ES2RADN Earth Radiance Present.** Indicates if earth radiance is present in ESA(B) north scan.

— **ES2RADS Earth Radiance Present.** Indicates if earth radiance is present in ESA(B) south scan.

- **Earth Present.** The following outputs indicate only if a scan has completely crossed the earth. Output is bilevel where 1 = Present, 0 = Not Present.

  — **ES1EPN Earth Present.** Indicates that ESA(A) north scan has crossed earth.

  — **ES1EPS Earth Present.** Indicates that ESA(A) south scan has crossed earth.

  — **ES2EPN Earth Present.** Indicates that ESA(B) north scan has crossed earth.

  — **ES2EPS Earth Present.** Indicates that ESA(B) south scan has crossed earth.

- **Pitch Angle.** Except where stated, the following pitch error angles from the ESA are telemetered as 8-bit two's complement numbers. The pitch range is ±5.12°.

  — **PES1N Pitch Angle.** Pitch angle from ESA(A) north scan.

  — **PES1S Pitch Angle.** Pitch angle from ESA(A) south scan.

  — **PES2N Pitch Angle.** Pitch angle from ESA(B) north scan.

  — **PES2N Pitch Angle.** Pitch angle from ESA(B) south scan.

  — **EPITCH Pitch Angle.** Pitch angle from selected ESA scan. This is an analog signal with a range of ±5.1° @ 2.008°/v.

- **Wheel Outputs.** The following sensor outputs relate to the reaction wheels.

  — **WHL1V Wheel Voltage - 1.** Voltage for Phase 1 of reaction wheel two-phase motor. Range 0 - 25 VAC.

  — **WHL2V Wheel Voltage - 2.** Voltage for Phase 2 of reaction wheel two-phase motor. Range 0 - 25 VAC.

  — **WHLI Wheel Current.** Reaction wheel motor current. Range: 0 - 7.2 Amps.

— **WHLDIF Dif Wheel Speed.** Wheel speed deviation from 3000 rpm. Range: ±612 rpm.

— **WHLCON Reaction Wheel Control.** Control signal to RWA motor electronics. Value dependent on whether wheel is caged or not. Range: Uncaged ±0.77°, Caged: ±153 rpm.

— **WHLT Reaction Wheel Temperature.** Temperature of the reaction wheel bearing. Range: -10° to +140° F.

Functional Description

A functional block diagram of the pitch control channel is illustrated in Figure 10, and pitch wheel controller characteristics are listed in Table 2.

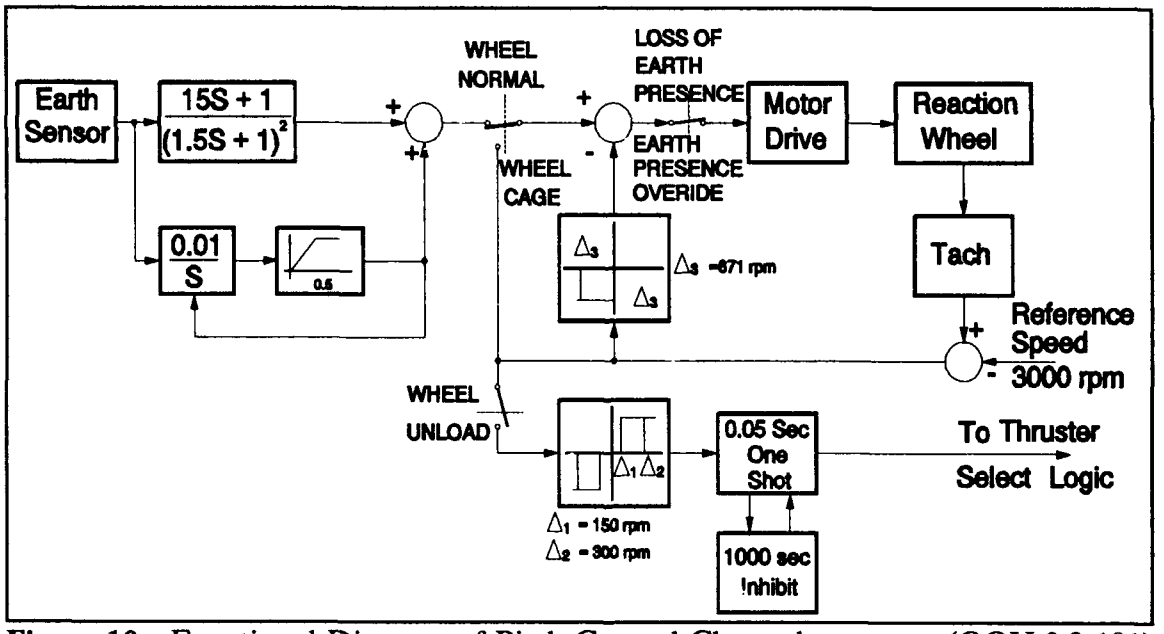

Figure 10 - Functional Diagram of Pitch Control Channel (OOH:3.2-101)

Normal Loop Operation. Pitch error voltage from the earth sensor is fed into a lead/double-lag compensator and an integrator circuit before being summed and sent to the motor drive electronics. The lead/double lag compensation derives rate for stability. The integrator prevents a constant error by providing the steady

state signal required to maintain motor torque equal to the wheel drag torque. If the drive torque were not compensated, there would be a constant attitude error equivalent to the wheel drag torque. Voltage proportional to the compensated signal drives both windings of the wheel motor, with the developed motor torque approximately proportional to the square of the compensated signal.

Table 2
Pitch Wheel Controller Characteristics (OOH:3.2-102)

Description	Characteristic
Maximum Motor Torque, Nominal (in-oz)	19
Wheel Drag Torque Range (in-oz)	0.6 to 2.5
Pitch Axis Moment of Inertia (slug-ft^2)	640
Pitch Attitude Error (deg)	< 0.05
Wheel Control Stability Margin: Gain (dB) Phase (deg)	 > 10 > 30
Momentum Removed per Unloading Sequence (ft-lb-sec)	0.35
Wheel Momentum Unloading Interval (days)	> 8

Wheel Momentum Unloading. During the wheel unload mode, the wheel speed is maintained within ±10 percent of 3000 rpm by comparison of the speed error with the 3000 rpm reference, and firing a pitch thruster when the threshold is exceeded. Pitch thruster firings are pulses of 0.05 second duration. These firings turn the spacecraft (about 0.25 deg/firing) and the resulting error is sensed and removed by the reaction wheel control, thereby adjusting the wheel speed. There is a 1000 second delay between pitch firings to ensure that the pitch signal is fully decayed before another firing. Under normal on-orbit operation, the wheel speed

is monitored and wheel unloading is accomplished manually. This manual unloading occurs approximately once every 20 days.

Wheel Run-up. On module power-up (WCE and WDE ON), wheel run-up is provided by the tachometer feedback when in the WHEEL CAGE ENABLE and EARTH PRESENCE OVERRIDE mode. With earth presence, the wheel will run-up because the tachometer loop is dominant over the attitude signal. When a speed of 3000 rpm is reached, the tachometer loop ceases to control the wheel and the attitude then commands the wheel to maintain null earth sensor pitch signals.

Wheel Cage. When WHEEL CAGE is enabled, the cage loop maintains the wheel speed at near 3000 rpm. This caged mode is used for velocity correction maneuvers and to run up the wheel without earth presence with earth presence override enabled. Normally the reaction wheel motor will be automatically disabled upon loss of earth presence.

Earth Sensor Assembly

The earth sensor assembly (ESA) uses an infrared detector to provide pitch and roll pointing data to the attitude control subsystem. Each ESA consists of a dual-scan sensor head subassembly and an electronics subassembly. The ESA heads are mounted on an accurately aligned pedestal structure. Each ESA head contains a scan mechanism and two infrared telescopes to provide two parallel scans, offset ± 5 degrees from the sensor boresight axis which is aligned with the spacecraft Z-axis. The scanning mechanism rotates a mirror to provide an optical scan length of ± 12.5 degrees. The mirror uses an optical incremental encoder to provide two outputs of angular readout. The first is a sequence of pulses, with pulses at equal angular increments, and the second is a reference pulse that occurs at the center of scan motion.

As illustrated in Figure 11, angle data from the encoder and the radiance data from the infrared detectors are fed into a computational logic circuit to produces pitch and roll error signals (10 bit serial data), and radiance and earth presence

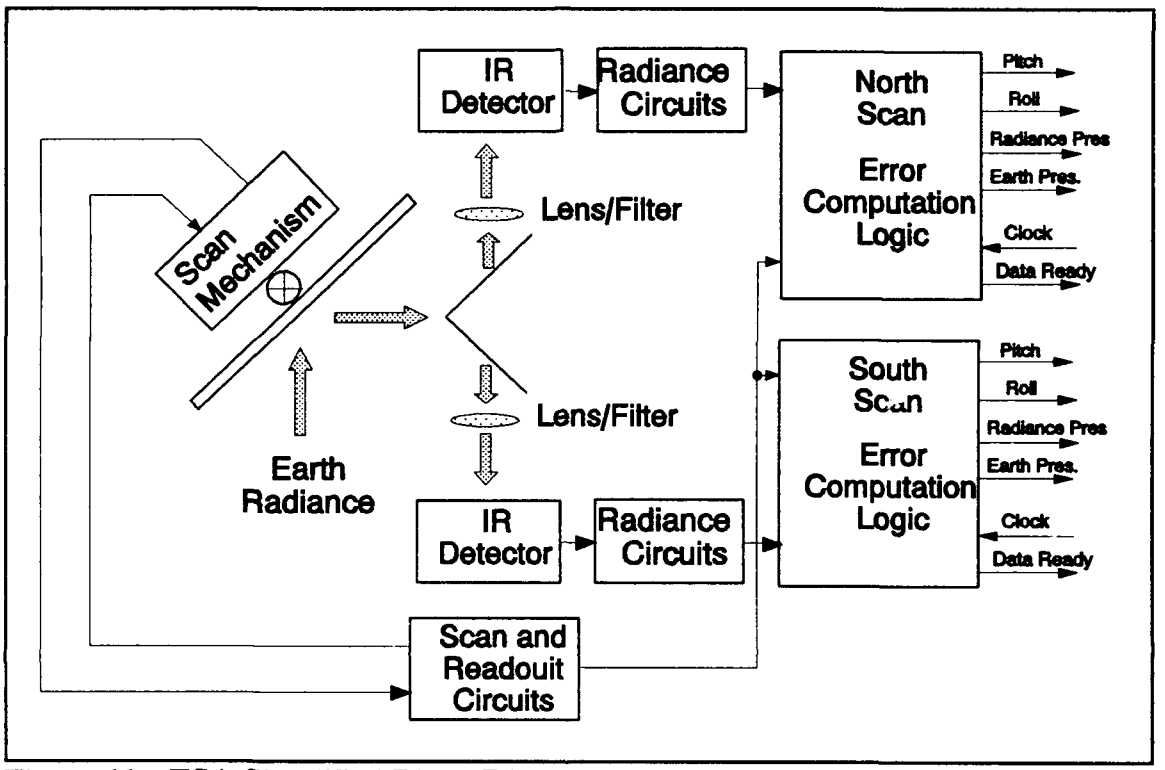

Figure 11 - ESA Simplified Block Diagram (OOH:3.2-139)

signals (bilevel). Although the ESA produces 10 bit serial data for pitch and roll error signals for the earth processing electronics, only 8 bit data (7 bit and sign) is telemetered. Each computation logic block outputs a Data Ready signal, and receives as input a clock signal, for clocking out the serial data. The pitch scale factor is 0.010°/count and the roll scale factor is 0.0071°/count. With a count of ±512 for the 10 bit word, the pitch range is therefore ±5.12 degrees.

As shown in Figure 12, the earth disk is scanned by two parallel scans, and each has a length of 25 degrees. From geo-stationary orbit, the earth diameter subtends an angle of about 17 degrees. With the Z-axis pointing at the center of the earth (perfect pointing), each beam receives radiance that subtends an angle of 14.7

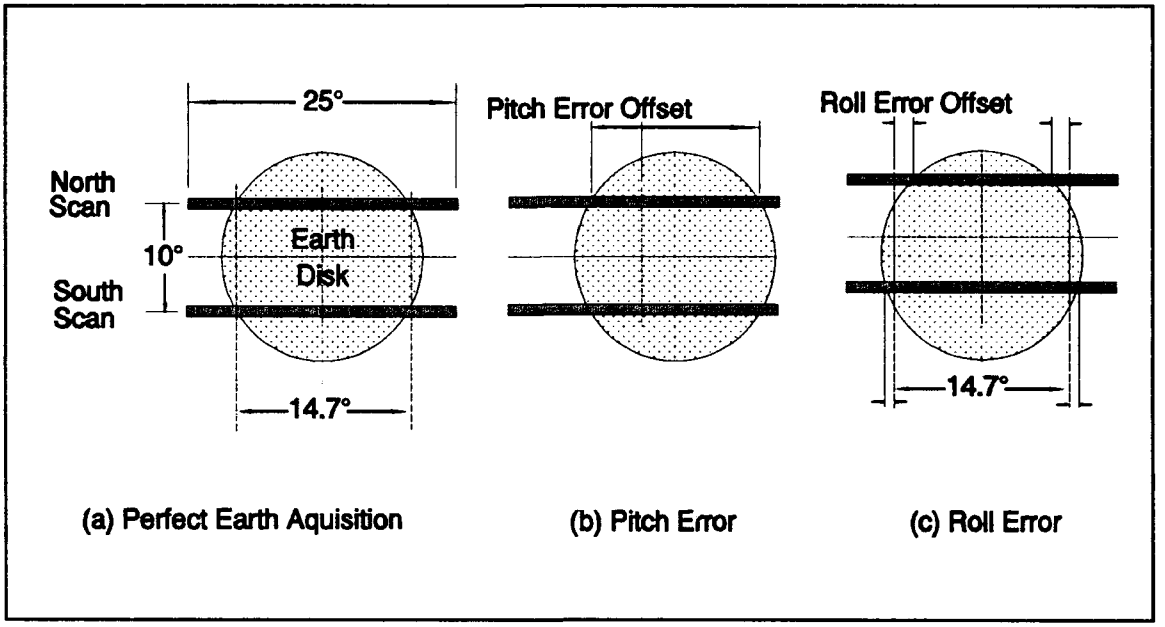

Figure 12 - Earth Scanning Beams

degrees across the earth disk. About five degrees of the scan overshoots each side of the earth disk. Perfect earth pointing scans are shown in Figure 12(a). Note that each scanning beam provides an independent measure of pitch and roll attitude error; the beams do not work together. The pitch error attitude is determined by the comparative lengths of the scanned half chords of the earth disk about an internal reference (Figure 12(b)). The roll attitude is determined by the total scanned chord length and compared against the length of the 14.7 degree reference chord (Figure 12(c)).

Earth Processing Electronics

The primary function of the Earth Processing Electronics (EPE) is to accept the pitch and roll error signals from the Earth Sensor Assembly (ESA) and convert the 10 bit digital words to analog form. The analog voltages are used to drive control circuits that correct the attitude of the spacecraft. The EPE provides the following four functions:

- Provide data-read clocks to the earth sensor in response to a data ready pulse.
- Select a specific earth sensor channel output in response to command inputs.
- Provide digital to analog conversion of the data.
- Provide sequential telemetry readout of all earth sensor data.

A simplified block diagram of the EPE is illustrated in Figure 13. The telemetry function plays no part in the functional operation of the pitch control channel and will not be further discussed.

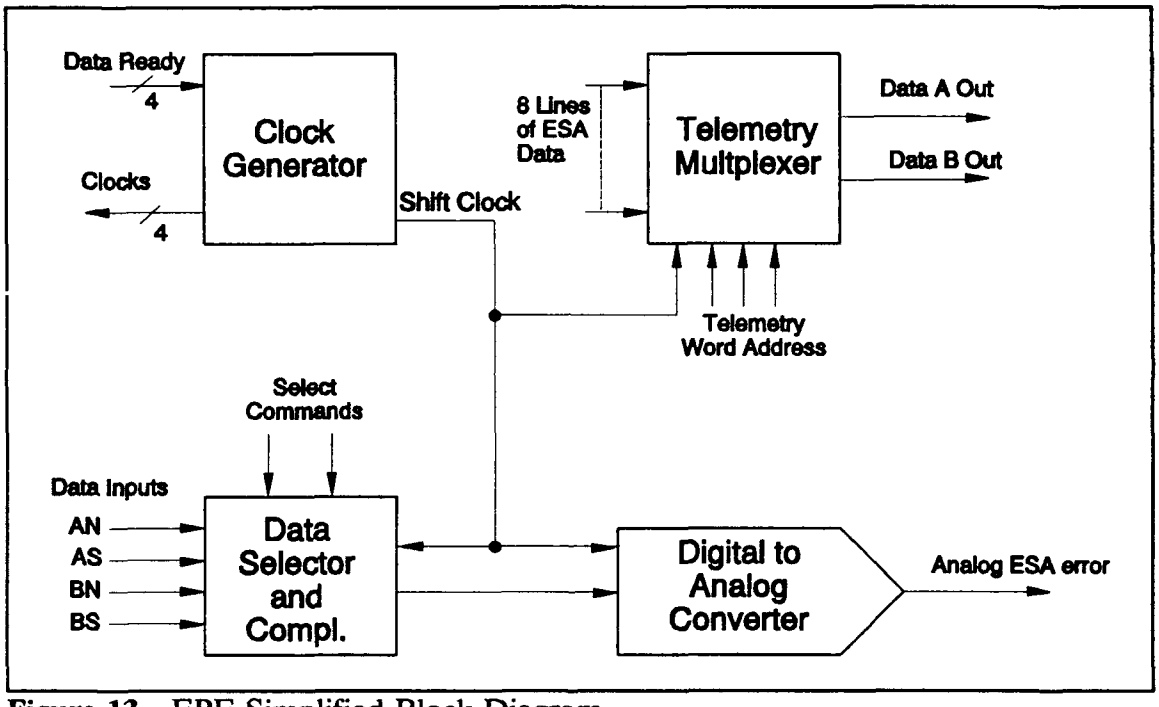

Figure 13 - EPE Simplified Block Diagram

Clock Generator. The clock generator circuit receives the asynchronous data read pulses from the ESA and responds with ten 9.6kHz data shift clock pulses. Although derived from the same phase of the dual phase master clock, two independent circuits are used to generate the shift clocks for the A and B ESAs. The second phase of the master clock is used in the rest of the EPE to carry out other timing functions.

Data Selection and Complementer. The data selection circuit uses a 4-line to 1-line multiplexer to select the required ESA scan. The required scan is determined by decoded select commands, telemetered from the ground. The circuit complements the pitch data from the north and south scan of the A ESA, and the roll data from the north scan of both ESAs. Complementing the data bits effectively reverses the polarity of the voltage from the digital to analog converter. The A and B sensor assemblies are physically mounted 180 degrees apart, and scan in opposite directions. Complementing is therefore necessary to ensure that the reaction control system applies the correct direction of torque to move the spacecraft to null the attitude error.

Digital to Analog Conversion. The digital to analog converter (DAC) receives the 10 bit serial bit stream from the data selector, stores it in a 10 bit receiving register, and converts the number to an analog voltage between -5 and +5 volts. The shift clock from the clock generator is used to shift the 10 bits into the receive register. A 6.4 volt reference voltage ensures that the digital to analog converter operates about a bias voltage of zero volts. An output buffer amplifier converts the digital to analog current output to a voltage with a scale factor of 9.77 mV per count. Another input to the DAC is the "zero register" signal. This signal forces all zeros into the 10 bit register when earth presence is lost, and prevents possible erroneous roll and pitch data forcing an unwanted attitude change. With an input scaling factor of 0.01°/count from the ESA, and the DAC output scaling factor of 9.77 mV per count, the voltage change per degree of pitch error angle is (9.77mV/cnt)/(0.01°/cnt), or 0.977 volts/degree. The transfer function could be expressed conversely as 1.024 degrees/volt.

Wheel Control Electronics

The main function of the Wheel Control Electronics (WCE) is to accept the analog earth sensor pitch error signals from the EPE, apply loop compensation and generate a wheel control voltage that is applied to the Wheel Drive Electronics (WDE) to control the speed of the reaction wheel. The WCE provides the following functions:

- Process the earth sensor pitch error signals and develop a wheel speed control voltage.
- Provide a "wheel cage" mode in which a tachometer pulse train is used to generate a wheel control voltage to drive the wheel speed to 3000 rpm.
- Provide wheel speed threshold detectors which cause thruster firing commands to be sent to the thruster valve drivers when the wheel speed is above or below set speed limits (wheel unloading).

The wheel cage and the wheel unloading play no part in the operation of the pitch control loop and will not be further discussed.

The function of the WCE is very important to the proper operation of the pitch control loop and has already been briefly discussed under Normal Loop Operation on page 56. Any closed-loop control system requires a filter that provides the correct damping on the system. If the filters were not correctly designed, the system could be underdamped or overdamped. Underdamping causes the system to overshoot the desired state, and in correcting for this, to overshoot again. It is hoped that the overshoot will become smaller with each excursion and the system will eventually reach its desired state. However, this can take an excessively long time. If the excursions do not become smaller, the system is unstable and continues to oscillate. Overdamping results in excessive time to reach the desired state; it may not reach it at all, leaving a residual error. Critical damping causes the desired state to be

reached in minimum time without any overshoot. This is usually the design goal, but often a little underdamping is designed into the system because the desired state can be reached in a slightly shorter time than with critical damping, but at the expense of a small overshoot.

The loop filter design parameters, which provide for the correct amount of damping, are determined from way the satellite mass reacts to a speed change in the momentum wheel. Engineers have determined that a lead-lag filter in parallel with an integrator stage can provide the optimum damping.

As shown in the simplified schematic of Figure 14, the pitch error processing circuit consists of an input buffer, a lead-lag filter stage, a parallel integrator, and an output summing amplifier. The buffer provides an inverting gain of 2.67. The buffer output is fed to two parallel paths: the lead-lag filter and the integrator.

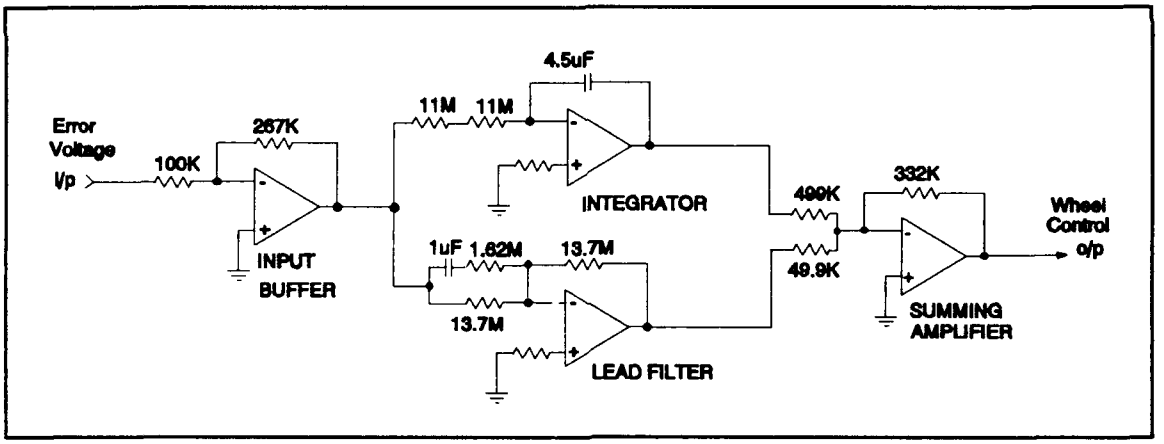

Figure 14 - Pitch Error Processing Circuits (OOH:3.2-195)

The lead-lag filter provides a transfer function of

$$H(s) = \frac{-(15.3s + 1)}{(1.37s + 1)(1.62s + 1)} \approx \frac{-(15s + 1)}{(1.5s + 1)^2} \quad .$$

The integrator circuit path provides a transfer function of

$$H(s) = \frac{-K}{Ts + 1} \quad ,$$

where

$$K \geq 75, \quad T \geq 7500, \quad \frac{K}{T} = 0.01 \pm 10\%$$

$$\therefore H(s) \approx \frac{0.01}{s} \quad .$$

The two paths are summed together in the output summing amplifier, which provides a gain of 6.65 to the lead-lag filter and a gain of 0.665 to the integrator output. The summing output is fed to the Wheel Drive Electronics (WDE) as the wheel control voltage.

Wheel Drive Electronics

The main function of the Wheel Drive Electronics (WDE) is to accept the control error voltage input from the Wheel Control Electronics (WCE) and use it to accelerate or decelerate the reaction wheel. This will produce torques that provide stability of the spacecraft about the pitch axis (Y-axis). The control loop includes four sections: the control error processing circuits, a pulse width modulator, a switching regulator, and the two-phase logic and drive amplifier circuits. These four blocks are illustrated in Figure 15.

Control Error Processing Circuits. The first block in the control error processing circuit buffers the wheel control voltage and converts any negative voltages to positive values. Hence, the output is the absolute value of the input. This absolute value is required because the motor speed control circuitry can only be driven with a positive voltage. This block also produces a bilevel polarity signal that

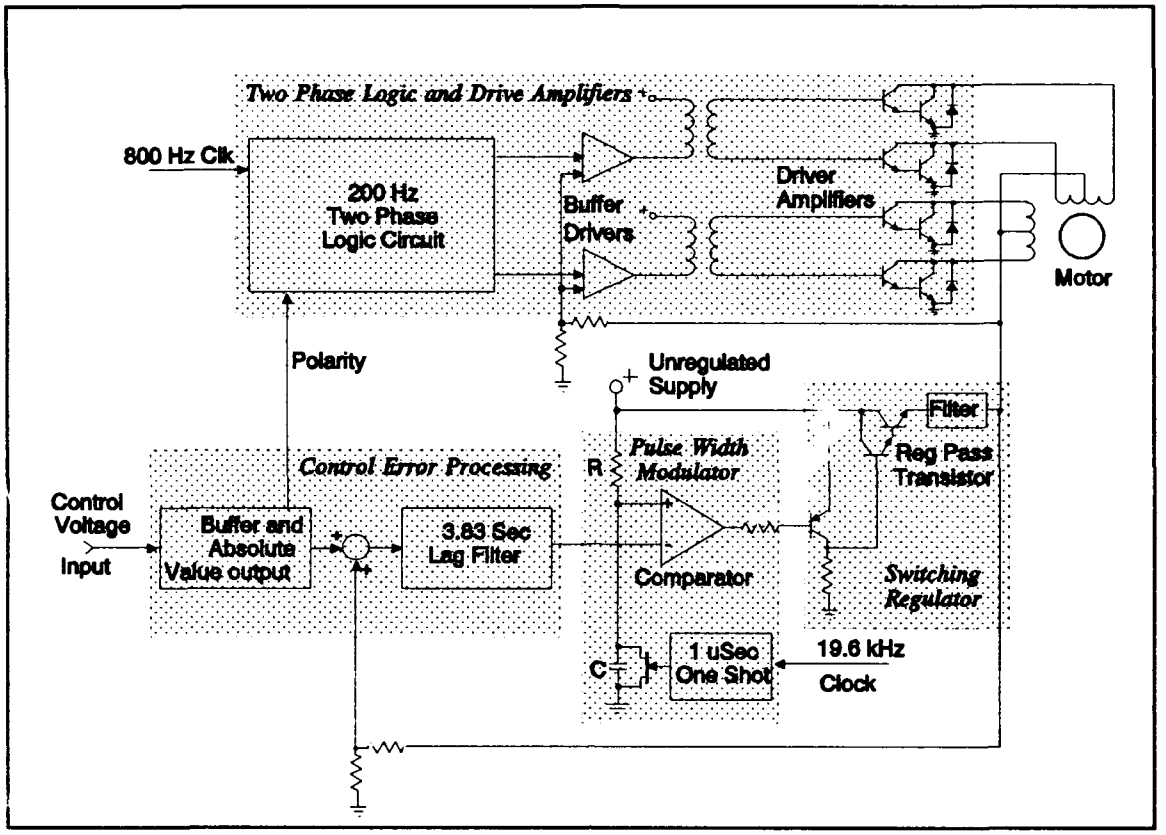

Figure 15 - Wheel Drive Electronics

is derived from polarity of the input wheel control voltage. This polarity signal controls the phase of the drive to the motor, and hence its direction. The absolute voltage is summed with a feedback voltage from the switching regulator and fed to a 3.83 second lag filter for loop compensation. The output of this filter drives the pulse width modulator which sets the duty cycle of the switching regulator.

Pulse Width Modulator. The pulse width modulator circuit consists of a comparator, an RC circuit and a 1 microsecond one-shot multivibrator. On the leading edge of the 19.6 kHz clock, a 1 microsecond pulse discharges the capacitor via the FET switch. The capacitor then charges to the unregulated supply voltage through R. The capacitor voltage is connected to the comparator positive input, and the wheel control voltage from the lag filter is connected to the negative input. The capacitor voltage eventually reaches the level of the wheel control voltage and the

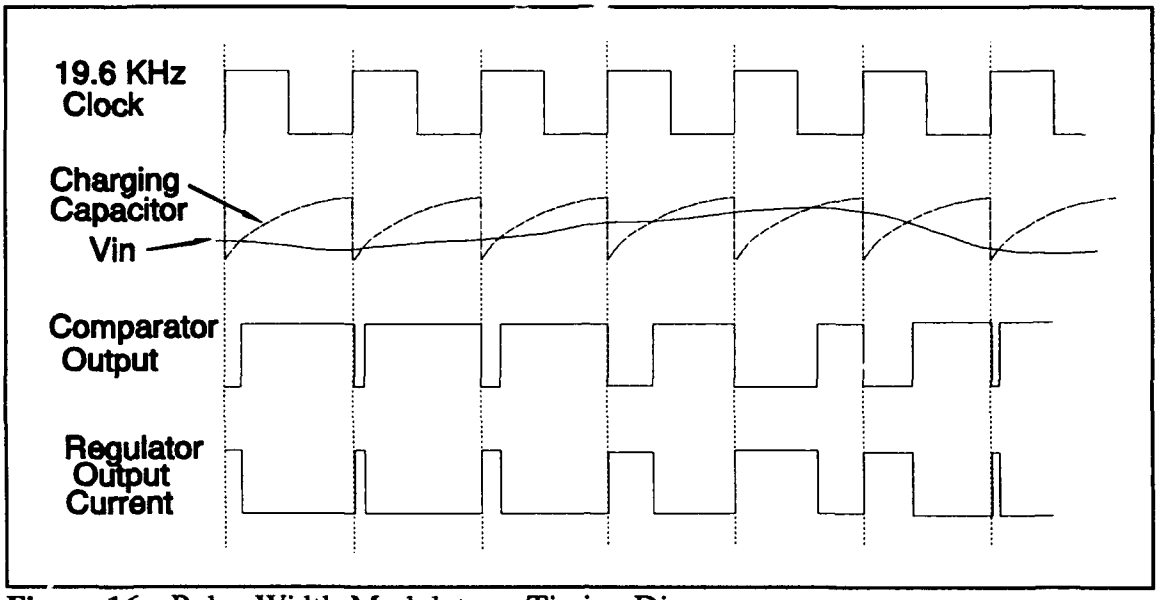

Figure 16 - Pulse Width Modulator - Timing Diagram

comparator output goes to the high state for the remainder of the 19.6 Khz clock period. This high state switches "off" the regulator output current. The comparator low output state switches the regulator output current "on". With high wheel voltage input, the regulator output current is on most of the time (of the 19.6 Khz clock period), and with low wheel voltage input, the regulator output current is off most of the time. Pulse width modulator timing is illustrated in Figure 16.

Switching Regulator. The filtered switching regulator output is connected to the center taps of the motor windings. The amplitude of the average current determines the speed of the motor. The comparator output drives the base of a PNP transistor, which in turn drives the base of a high current Darlington pair configured as a regulator series-pass transistor. The pulse-width-modulated current from the series-pass transistor is fed through an LC filter network to smooth out the pulses before being applied to the motor winding center taps.

Two Phase Logic and Drive Amplifier. The reaction wheel drive motor is a 200 Hz, two phase induction motor. Drive current for the motor is supplied from the switching regulator connected to the center tap of the motor windings. The ends of

the windings are switched to ground, with high current Darlingtons, in a 90 degree, two-phase sequence to provide for torquing in either direction. Inductive motor currents are shunted to ground by reverse-connected diodes across each driver transistor. The inputs of the driver amplifiers are transformer-coupled to buffer drivers which are driven by 200 Hz, two-phase logic circuitry. The drive current to the transformers consists of a fixed value, from the two-phase logic circuitry, and a variable value that is proportional to the switching regulator output. This ensures that the transistor base current is increased only when needed. The two-phase logic circuits derive their 90 degree, 200 Hz signals from a 800 Hz synchronous counter. The output of this counter is gated with the polarity signal, from the control error processing circuits, to switch the phase when required.

Reaction Wheel Assembly

The reaction wheel is an inertia wheel that produces corrective torques about the pitch axis (Y-axis) of the spacecraft whenever the wheel speed is accelerated or decelerated. During normal on-orbit operation, wheel speed is controlled by the wheel drive electronics (WDE) which receives the pitch-error voltage from the earth sensor assembly (ESA) via the earth processing electronics (EPE). The wheel spins at a nominal bias speed of 3000 rpm. The corrective torques maintain the spacecraft's pointing to the center of the earth about the pitch axis to within 0.25 degrees. Gyroscopic coupling of the wheel bias momentum with the spacecraft orbit rate provides passive control of the spacecraft about the yaw axis (Z-axis).

Wheel Description. The wheel described in the OOH has an outside diameter of 13 inches and weighs about 12.5 pounds. The two-phase induction motor is press-fit into the hub of the wheel. An electromagnetic pickup is mounted adjacent to the rim of the wheel, where a notch in the rim causes it to generate an electrical pulse

once every revolution. The temperature of the wheel bearing is sensed by a closely located thermistor.

Wheel Operation. The wheel drive electronics converts the pitch error signal into two-phase, 200 Hz, squarewaves which are applied to the induction motor inside the wheel. Depending on the phasing between the two signals, the wheel will be accelerated or decelerated providing the reaction torque to the spacecraft in a direction to reduce the earth pointing error. Whenever the wheel speed goes outside its limits, or when the wheel is not required for normal on-orbit operation (e.g., a spacecraft velocity change), the wheel is caged and its speed brought back to the nominal 3000 rpm. This is done by a closed loop circuit employing the WCE, WDE and the tachometer output.

Gyroscopic Coupling. The wheel spin axis is lined up with the spacecraft Y-axis, which is parallel to the orbit axis. If the spacecraft rotates about its yaw axis, away from its nominal position, the bias momentum of the wheel, coupled with the spacecraft's orbit rate, will provide gyroscopic torques to return the spacecraft to its proper orientation about the yaw axis.

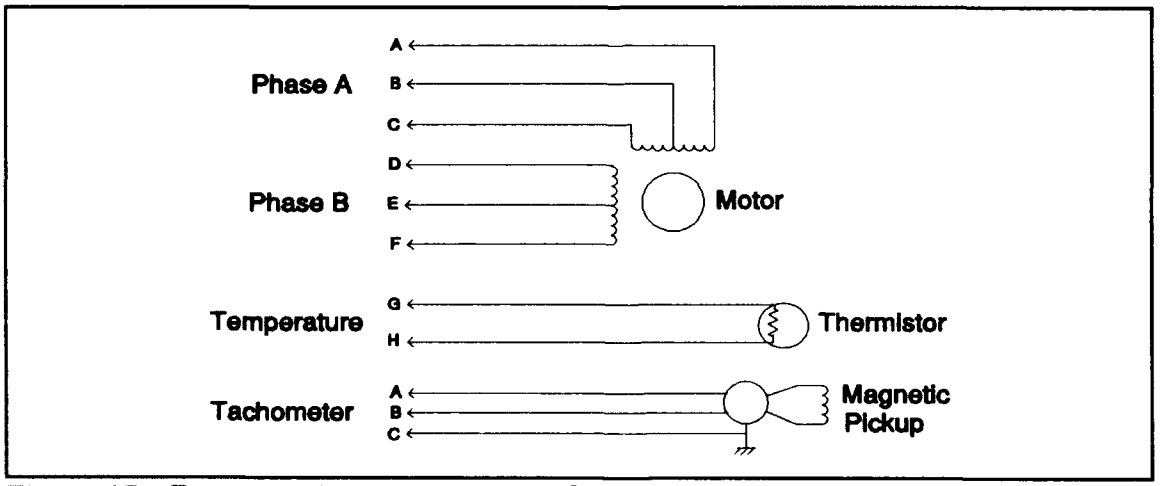

Figure 17 - Reaction Wheel Assembly - Schematic (OOH:3.2-268)

A schematic of the reaction wheel assembly is illustrated in Figure 17, and a list of performance parameters is listed in Table 3.

Table 3
Reaction Wheel Assembly Parameters (OOH:3.2-269)

Parameters	Performance	
Angular Momentum at nominal speed	6.77 ft-lb-s ±2.0%	
Nominal Speed	3000 rpm	
Speed variation range	±500 rpm	
Wheel run-up time	76 sec with 120 W or 25 V max applied	
Wheel torque (in-oz), 120W or 25V min: 2500 - 3317 rpm 3317 + 3565 rpm	Accelerating 15 10	Decelerating 14.5 10
Steady state power	12 W max at 70°F or higher increasing linearly with decreasing temp to 24 W at 0°F	
Excitation: Voltage Frequency Phases Phase shift	 24 to 30 V p-p, squarewave 200 ±1.0 Hz 2 90 ±5°	
Tachometer output: Polarity (pin B - Pin A) Amplitude at 3100 rpm: positive negative Slope between 1.2 and 0.8 V	 + for counter clockwise rotation 3.0 to 12.0 V 0.2 to 1.0 V -3000 V/s min	

Summary

The aim of this chapter was to present a part of a satellite subsystem that can be modelled for use in an automatic fault detection system employing model-based reasoning. The attitude velocity control subsystem (AVCS) of a typical geo-stationary satellite was selected to be the subsystem for experimentation. An overview of the function of the AVCS was given, but due to complexity, only the pitch channel of the

AVCS was presented in enough detail to enable computer modelling for the model-based reasoner to operate on.

This chapter first presented a general description of the satellite, followed by a description of the AVCS. The AVCS's hardware can be configured in one of seven modes, corresponding to the function of the satellite at any given time. Configuration in the 4B-Normal mode was selected because that is the mode the satellite is in for most of its life. Only the pitch control channel, in the 4B mode, was presented in detail; all inputs, outputs, and a functional description, for the channel as a whole, were given. Individual components of the channel (those that can be switched to a redundant backup) were then described in enough detail to enable computer modelling.

V. Software Development

Introduction

In an autonomous satellite, fault detection software would become part of an imbedded system and tightly integrated with the hardware. Since the software is intended to function automatically, the software can be less complex because there is no human interface. As a real satellite's telemetry data was not available for software validation, a simulation of a real system was used to test the software. The test facility will of course, require at least a rudimentary human interface. To save programming effort, the rudimentary interface is provided by the basic input/output facilities of the development language itself.

In this chapter, the criteria for selection of a computer language are discussed. The computer model for the pitch control channel is developed by first defining the objects and their behavior, then their interconnection structure. This is followed by the development of a model-based diagnostic reasoner. The reasoner is based on Scarl's work, as the fault detection problem is similar to a process control system with commands and many sensors.

Language Selection

Several computer languages were investigated with the potential to accomplish this programming task. A list of desired characteristics for the programming language (listed in priority order) is shown below:

- **Object-Oriented.** Each component has its own input, output and function. Each component is an object whose internals are manipulated only by sending messages to and from the object. Object-oriented languages are designed to do exactly this,

with the added benefit of fast prototyping and protection against inadvertent modification of data inside objects

- **Common Language.** A common language that is portable across machine types. Common refers to a language that many programmers know, and portability means compilers exist for a number of processors. Ultimately, this will lead to faster, and less costly development.
- **Convenience.** The language should provide convenient programming features. For example, it should be able to manipulate symbolic data by its name. The language should not burden the programmer with taking care of housekeeping chores, such as memory allocation and de-allocation. Convenience also means the language should provide good debugging facilities. These conveniences enable rapid prototyping, which again will lead to a faster, cheaper and maintainable product.
- **Efficient Code.** The language should be able to produce tight and efficient machine code. For on-board satellite operation, computer resources are always at a premium. To fit most functionality into the smallest space (rom, ram, disk space, and the physical hardware) code size and efficiency is important.
- **Software Development.** The language should enable software development on a personal computer (PC). PCs are a relatively inexpensive resource. As PCs and compilers become more sophisticated, many programmers will move their software development to these platforms. In the long term, this can result in considerable saving in software development expense. The saving is not only because PCs are cheaper, but also because many programmers have access to these machines and their cheaper (but not less sophisticated) development tools.

The languages considered in this study were, Smalltalk, Lisp with Flavors, C++, and Scheme with SCOOPS. *Flavors* and *SCOOPS* are object-oriented

extensions to the Lisp and Scheme languages respectively. As with any engineering endeavor, the characteristics listed above cannot all be met and typical engineering tradeoffs must be made. The ideal language is one that provides a very high level of abstraction, can handle symbols and manipulate objects without any consideration of the overhead, and compile code with the efficiency and compactness of an assembler.

This programming task entails building a model that can be simulated in the computer, then building the routines that manipulate that model to carry out model-based reasoning to diagnose faults in the model. Because of the simulation nature of this task, object-oriented programming (OOP) is almost essential, if not ideal, for this task. All of the four languages listed are capable of OOP. Smalltalk, an excellent OOP language, was rejected because its user interface is human oriented, and consequently has too much overhead. Lisp with Flavors, another excellent development platform requires the use of a specialized Lisp machine for efficiency. It would be very difficult to efficiently transfer Lisp code to an imbedded on-board system. C++ is a reasonably high level language, produces tight code and its software can be developed on PCs. C++ can also be compiled (or translated through C) on a variety of platforms to produce machine code that may be ROMable, and so suitable for an on-board system. Scheme is a high level language, similar to Lisp but much simpler and so does not require Lisp's overhead. Scheme programs can be developed on a PC, but it does not operate as a true compiler, and so does not produce native machine code that is ROMable.

Initially, the language chosen was C++, but after four weeks of experimentation, and encountering problem after problem, this was abandoned in favor of Scheme. The C++ development system evaluated was Borland's Turbo C++. It is a higher-level language than C, but its level is still far below that of Lisp or Scheme. The programmer is responsible for many mundane tasks, such as

memory allocation for objects and other variables. Lists and arrays of objects were created but they could not easily be manipulated as symbols. Some problems were also experienced with corruption of encapsulated data. That is, data inside an object (such as the voltage output state of an amplifier object) should be changeable only by sending a message to the object (amplifier to calculate its output given its inputs); however, it changed by some other means, unknown to the author. Perhaps the biggest problem with the C++ language was this author's lack of experience with C and not mastering the C++ extensions in the short time available. This author still believes C++ to be the best programming language for the task, provided the final product has to be installed into an on-board system. Borland's implementation certainly has the potential for this because of its efficient code generation and its superb user interface that provides for convenient software development, code optimization and debugging. Not having time to pursue C++, the author chose PC Scheme for this programming task because of its simplicity, superior symbol manipulation, PC usability, and most importantly for this thesis effort— fast prototyping.

The Scheme Programming Language

Scheme is a dialect of Lisp. It is simpler than Lisp but really not less powerful. Lisp achieves its power by providing a vast array of ready made functions for the user. In Scheme, many functions have to be built by the programmer. An important difference between older Lisp dialects and Scheme is that Scheme is *lexically scoped* and all data types and structures are *first-class objects*. These two differences affect the way a programmer uses the language. Lexical scoping means each variable only exists inside a function call (procedure), and never outside the function. This provides a degree of protection for variables inside a function (often called data encapsulation, or information hiding) and is an essential part of object-

oriented programming in Scheme. Programmers benefit because they do not have to be as concerned about naming their variables and it encourages a functional programming style. Having all data treated as first-class objects means all data are treated in a consistent manner. That is, complex data types such as structures, arrays and functions themselves, can be passed to functions as easily as can simple data types. This consistency, in treating all data as first-class objects, simplifies programming considerably because the programmer does not need to know in advance what can, and what cannot, be manipulated in certain ways.

This software design project uses Texas Instruments' PC Scheme, and its OOP extension called SCOOPS, an acronym for SCheme Object Oriented Programming System. The concepts and syntax of SCOOPS are similar to those of other Lisp based object-oriented extensions such as LOOPS and Flavors; they rely, however, on features of the Scheme language discussed above.

Object-oriented programming manipulates objects which represent abstract entities. An object is composed of variables and methods. Variables determine the local state of the object, and methods are procedures or functions that define the object's behavior. Higher levels of abstraction are built up through inheritance. That is, higher-level classes of objects may inherit the properties of other classes. Large systems can therefore be built from parts that can be developed and maintained separately. The only way to interact with an object is by sending a message to it. An exception: methods that belong to a particular object can manipulate that object's variables directly.

Software Development

The software was developed in three phases. The first phase develops the objects used in the model. The second phase develops the structure, the way the objects are interconnected. The third phase develops the model-based reasoning mechanism to diagnose faults in a real system using the model. A computer simulation of a real system will be used to validate the reasoning mechanism. Therefore, there will be two computer models, one representing the real system which can be "broken", and the other that is used to generate expected values for the "real" sensors used in the diagnostic reasoning.

Function

The objects representing components in the pitch control channel, will be descendants from higher class objects, or super classes that contain attributes common to all objects. The super classes will be developed first, followed by the individual objects in the pitch control channel. Development order will be: the Earth Processing Electronics (EPE), the Wheel Control Electronics (WCE), the Wheel Drive Electronics (WDE), the reaction wheel itself (WHEEL), and the Earth Sensor Assembly (ESA). Although the pitch error signal emanates from the ESA, the ESA object will be used to model the spacecraft's reaction to wheel acceleration. This way, the rate of pitch error reduction can be calculated. The SCOOPS class definition is used to declare an object's variables. The object's function is defined in the defined classes methods. Class definitions for all components, and their methods are listed in file PDEF.S at Appendix A.

Super Classes

All objects have several things in common. Therefore a super-class called COMPONENT is created that includes all the common attributes of the objects in the pitch control channel. For example, each object has a name, a status, input and output lists (that hold a list of objects connected to the input and output respectively), and a state (calculated from its inputs) that is transmitted to those objects in the output list. An example SCOOPS class used to define COMPONENT is shown in Listing 5. Note that a class only *defines* the variables and methods an

```
(define-class COMPONENT
              (instvars
                name
                (status  'on)
                (state   0)
                (input-list '())
                (output-list '()) )
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class COMPONENT)
```

Listing 5 - SCOOPS Example for COMPONENT Class

object can use. It does not create the actual object that can be connected into a circuit. An object is created and can be connected into the circuit when an "instance" of the class is defined. This will be done under the Structure development below. The example in Listing 5 highlights some important points. Instance variables are created using the *instvars* keyword. The *name* variable is on a line by itself. This means it is initially unassigned and only gets filled when an object is instantiated. The *status* variable stores the status of the object; in this case, it is initialized to "on". The *state* variable stores the output state of the object and is shown initialized to 0. This variable is only set by the *deposit-value* method, which recursively propagates the value of *state* through the system, starting with connected objects declared in the *output-list*. The deposit-value method is shown in Listing 6.

```
(define-method (COMPONENT deposit-value)
  (value)
  (set! state value)
  (tell-other output-list) )

(define (tell-other output-list)
  (cond
    ( (null? output-list)
      nil )
    (else
      (send (eval (car output-list)) update)
      (tell-other (cdr output-list)) )))
```

Listing 6 - Example Method for COMPONENT Class - Deposit-Value

The *input-list* and the *output-list* are both initialized to an empty list. The gettable-, settable-, and inittable-variables statements ensure that each of the instance variables listed, can be got, set and initialized respectively. In SCOOPS, all class definitions follow this basic arrangement. Using the *describe* function of SCOOPS with the class name as an argument, displays the class as shown in Listing 7. As can be seen in the Figure, the description not only lists the methods defined by the

```
[6] (describe component)

    CLASS DESCRIPTION
    =================

 NAME             : COMPONENT
 CLASS VARS       : ()
 INSTANCE VARS    : (NAME STATUS STATE INPUT-LIST OUTPUT-LIST)
 METHODS          : (DEPOSIT-VALUE SET-NAME SET-STATUS SET-STATE SET-INPUT-LIST
SET-OUTPUT-LIST GET-NAME GET-STATUS GET-STATE GET-INPUT-LIST GET-OUTPUT-LIST)
 MIXINS           : ()
 CLASS COMPILED   : #T
 CLASS INHERITED  : #T
()
```

Listing 7 - SCOOPS Class Description for COMPONENT

programmer (deposit-value), but also those methods generated automatically by SCOOPS that get, set and initialize variables in the *INSTANCE VARS* list.

Another class that can be considered a super-class, because it is used to derive other classes and not used to instantiate working objects, is the AMPLIFIER class. The structure of this class is shown in Listing 8. The *(mixins COMPONENT)* line shows that this class is inherited from the COMPONENT class. All variables and

```
(define-class AMPLIFIER
              (instvars
                gain
                limit
                tolerance
                (fault-list '(high low zero latchup latchdown)) )
              (mixins COMPONENT)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class AMPLIFIER)


(define-method (AMPLIFIER update)
  ()
  (case status
    ( 'on         (let ( (vo (amp-state)) )
                    (if (number? vo)
                        (deposit-value vo)
                        nil )))
    ( 'off        (deposit-value nil) )
    ( 'latchup    (deposit-value limit) )
    ( 'latchdown (deposit-value (* limit -1)) )
    ( 'zero       (deposit-value 0) )
    ( 'high       (let ( (vo (amp-state)) )
                    (if (number? vo)
                        (deposit-value
                          (let ( (hi-val (+ vo 5)) )
                            (if (>? hi-val limit)
                                limit
                                hi-val )))
                        nil )))
    ( 'low        (let ( (vo (amp-state)) )
                    (if (number? vo)
                        (deposit-value
                          (let ( (lo-limit (* limit -1))
                                 (lo-val (- vo (* vo 0.2))) )
                            (if (<? lo-val lo-limit)
                                lo-limit
                                lo-val )))
                        nil )))
    (else (writeln "*** ERROR: Invalid status: " status)) ))


(define-method (AMPLIFIER amp-state)
  ()
  (let ( (vi (send (eval (car input-list)) get-state)) )
    (if (null? vi)
        nil
        (let ( (vo (* gain vi))
               (upper-rail limit)
               (lower-rail (* limit -1)) )
          (cond ( (or (in-range? state vo tolerance)
                      (and (>? vo upper-rail)
                           (equal? state upper-rail) )
                      (and (<? vo lower-rail)
                           (equal? state lower-rail) ))
                  nil )
                (else
                  (cond
                    ( (>? vo upper-rail)
                      upper-rail )
                    ( (<? vo lower-rail)
                      lower-rail )
                    (else
                      vo ))))))))
```

Listing 8 - AMPLIFIER Class

methods in the COMPONENT class are now available to all derived classes (or instances) of the AMPLIFIER class. The *gain* variable represents the gain of the amplifier, and the *limit* variable represents the positive and negative voltage swing limits of the amplifier output. The *tolerance* variable is required for analog objects to enable voltage comparison against a range rather than checking against an absolute value. For example, in a feedback circuit, signals will propagated through objects in a circular fashion. When the State variable comes within the tolerance range, propagation ceases, and the system can be considered stable. The *fault-list* includes a number of possible fault conditions that a typical amplifier can experience. These conditions are used by the model-based reasoning algorithm to diagnose the system. Most components in the system are essentially amplifiers. They have a gain, or some other transfer function, and will derive their classes and methods from the AMPLIFIER class.

As briefly mentioned above, each object has an *update* method that calculates the state of the object from the inputs. To determine the state for an amplifier whose status is "on" (no fault conditions), the update method calls the amp-state method which determines the output state by a simple multiplication of the gain with the input. The amp-state method also checks the new state against the limits and adjusts the state accordingly. If the amplifier's status is something other than "on", the update method determines the state accordingly. For example, if the status is "latchup", the update method will place the value of the positive limit in the state variable. The UPDATE and AMP-STATE methods have been listed together with the AMPLIFIER class definition in Listing 8. If a class derived from the AMPLIFIER class uses a method with the same name as one of AMPLIFIER's methods, then the new method will override the AMPLIFIER's method. In object-oriented programming parlance, this is often called *overloading*. As will be seen, most

objects will use AMPLIFIER's UPDATE method, but will overload the AMP-STATE method because that is where the output *state* is calculated from the input.

The SCOOPS description for the AMPLIFIER class is shown in Listing 9. Notice that most variables and methods are inherited from the COMPONENT class, shown here declared in the *MIXINS* line.

```
[7] (describe amplifier)

    CLASS DESCRIPTION
    =================

 NAME             : AMPLIFIER
 CLASS VARS       : ()
 INSTANCE VARS    : (OUTPUT-LIST INPUT-LIST STATE STATUS NAME GAIN LIMIT
TOLERANCE FAULT-LIST)
 METHODS          : (AMP-STATE UPDATE GET-OUTPUT-LIST GET-INPUT-LIST GET-STATE
GET-STATUS  GET-NAME  SET-OUTPUT-LIST  SET-INPUT-LIST  SET-STATE  SET-STATUS  SET-NAME
DEPOSIT-VALUE SET-GAIN SET-LIMIT SET-TOLERANCE SET-FAULT-LIST GET-GAIN
GET-LIMIT GET-TOLERANCE GET-FAULT-LIST)
 MIXINS           : (COMPONENT)
 CLASS COMPILED   : #T
 CLASS INHERITED  : #T
()
```

Listing 9 - SCOOPS Description for AMPLIFIER Class

Earth Processing Electronics

The earth processing electronics (EPE) uses a digital to analog converter (DAC) to convert the pitch error angle from the earth sensor assembly (ESA) to a voltage used by the wheel drive electronics (WDE). This simple operation can be

```
(define-class EPE
              (instvars
                (gain  0.976)
                (limit  5)
                (tolerance 0.001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class EPE)
```

Listing 10 - EPE Class Definition

modelled with an amplifier that has a gain equal to the transfer function of the DAC.

This transfer function was found in the last chapter to be 0.976 volts/degree. The output is limited to ±5 volts. The EPE class definition, which is derived from AMPLIFIER class, is shown in Listing 10. It defines no methods of its own, using all those of the AMPLIFIER class and the COMPONENT class.

Wheel Control Electronics

The wheel control electronics is more difficult to model than the EPE. It provides the loop compensation for the pitch control system, and is therefore a time-dependent object. From Figure 14 and the discussion in the last chapter, the transfer function of this object can be determined from the equations given for the lead-lag filter and the integrator. Since the integrator is required to overcome the wheel drag, and the wheel drag is not being modelled, the integrator can be ignored. The transfer function for the lead-lag compensation filter is given in the complex frequency s domain. The computer simulation is essentially a sampled-data, or discrete time, system. Therefore, a difference equation (the analog of a differential equation for continuous systems) will be used to model the filter. The difference equation is derived from the s-domain transfer function. To do this, the s-domain transfer function must first be transformed to the z-domain. The difference equation can then be derived directly from the z-domain transfer function.

Before deriving the difference equation, several assumptions must be made. First, that the WCE is a Linear Shift Invariant (LSI) system, so that s can be replace by $\frac{1 - z^{-1}}{T}$. Second, that the sample time T is one second. This could be enforced in an on-board diagnostic system by using a real-time clock and interrupt hardware to read sensors at one-second intervals, and having the computer model complete each loop through the circuit in one second. The one-second sample time assumes that the input varies slowly in comparison to the sample time. If this were not the case, aliasing errors would become a factor. The one-second sample time

should be a good assumption because, in reality, the satellite typically takes hundreds of seconds to correct for small (less than one degree) pitch errors. Given the sample time $T = 1$, transformation of the transfer function from the s to z domain can be accomplished using $1 - z^{-1}$.

A block diagram of the WCE showing the s-domain transfer functions in each block, is illustrated in Figure 18. Ignoring the integrator, and placing all gain

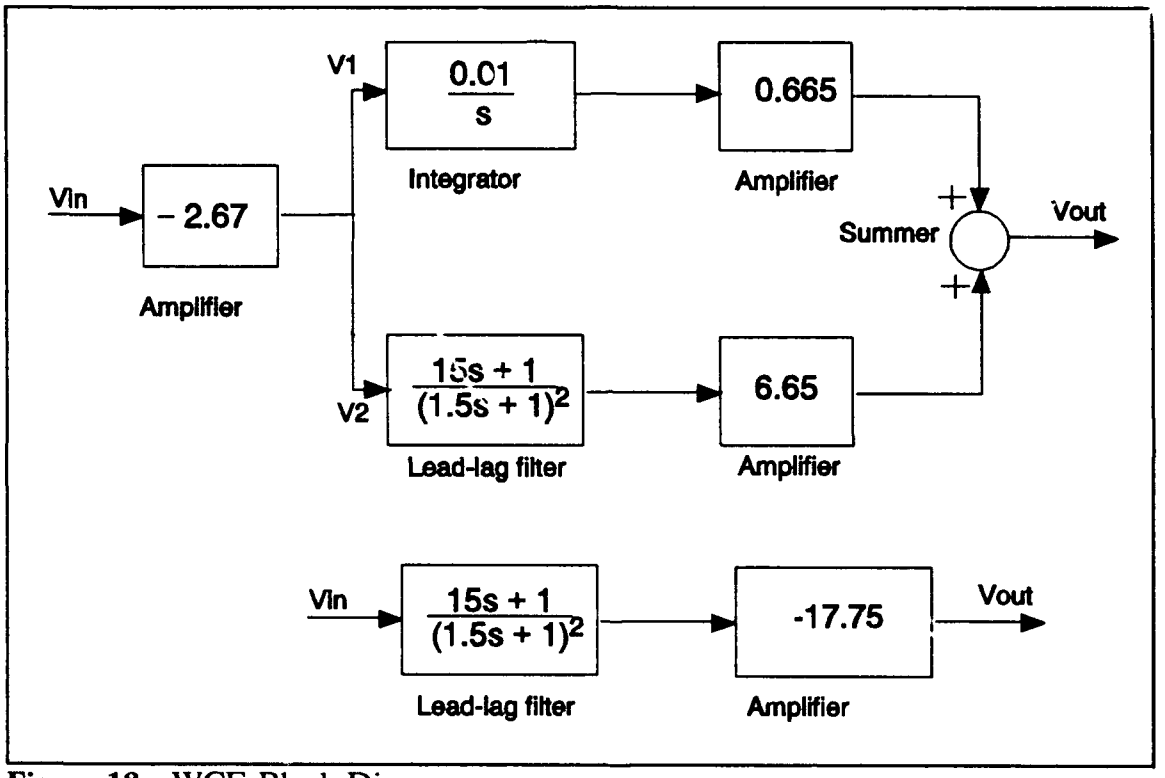

Figure 18 - WCE Block Diagram

components at the output, the lower block diagram in Figure 18 will used for this simulation.

Given the s-domain transfer function

$$H(s) = \frac{Y(s)}{X(s)} = \frac{15s + 1}{(1.5s + 1)^2} \quad ,$$

and substituting $1 - z^{-1}$ for s, and expanding, gives

$$\frac{Y(z)}{X(z)} = \frac{15(1 - z^{-1}) + 1}{[1.5(1 - z^{-1}) + 1]^2} = \frac{16 - 15z^{-1}}{6.25 - 7.5z^{-1} + 2.25z^{-2}} \quad .$$

A subsequent cross-multiplication, gives

$$6.25Y(z) - 7.5z^{-1}Y(z) + 2.25z^{-2} = 16X(z) - 15z^{-1}X(z) \quad .$$

To convert this equation to a difference equation, we use the following relations:

$$\begin{aligned} Y(z) &\rightarrow y(n) \\ z^{-1}Y(z) &\rightarrow y(n-1) \\ z^{-2}Y(z) &\rightarrow y(n-2) \\ X(z) &\rightarrow x(n) \\ z^{-1}X(z) &\rightarrow x(n-1) \quad . \end{aligned}$$

Substituting the above relations and rearranging in terms of the output y(n) gives

$$y(n) = 2.56x(n) - 2.4x(n-1) + 1.2y(n-1) - 0.36y(n-2) \quad .$$

This difference equation will be used in the model to determine the output of the filter. As can be seen, the calculation of y(n) not only requires the current input x(n), but also requires the previous input x(n-1), and the two previous output values, y(n-1) and y(n-2). Figure 19 through Figure 21 show the results of a Quattro™ spreadsheet simulation of this difference equation. The step input, in Figure 21, shows that the output is underdamped.

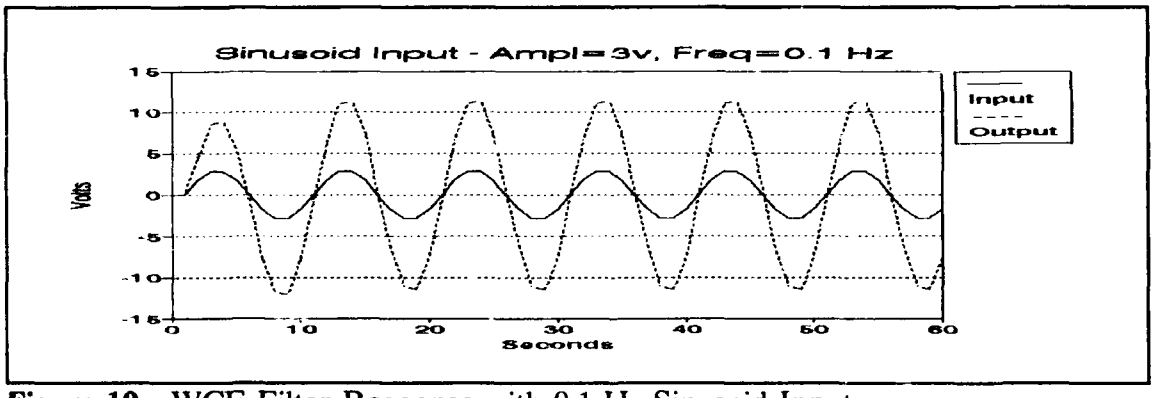

Figure 19 - WCE Filter Response with 0.1 Hz Sinusoid Input

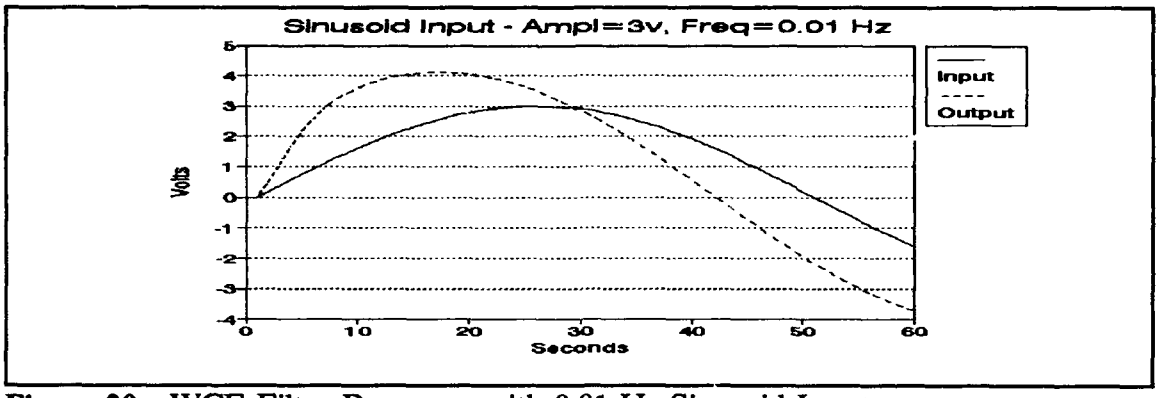

Figure 20 - WCE Filter Response with 0.01 Hz Sinusoid Input

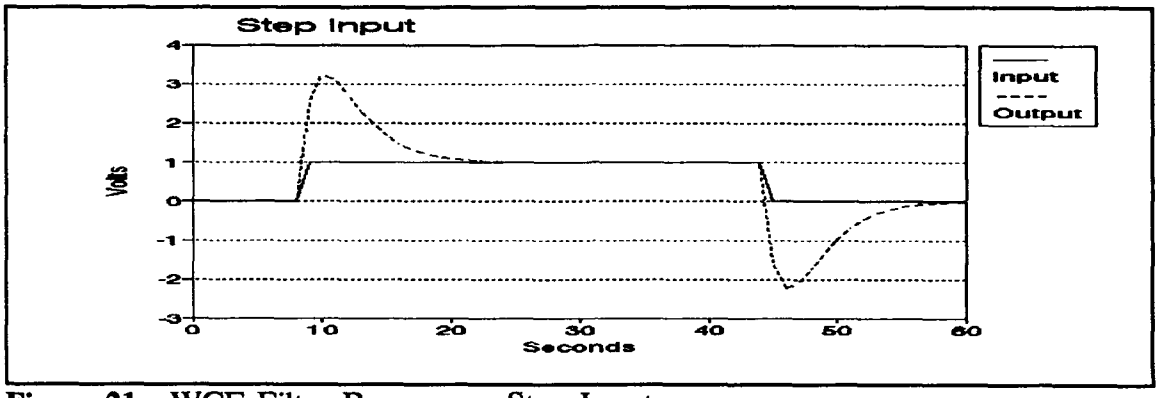

Figure 21 - WCE Filter Response - Step Input

The Output state of the WCE is given by the difference equation result, y(n) multiplied by the gain of the last stage. With an output swing limit of ±2′ volts, and a tolerance set to 0.001, the WCE class can be defined as shown in Listing 11.

Wheel Drive Electronics

The wheel drive electronics (WDE) takes the compensated control voltage from the WCE and turns it into a voltage, with a range of ±25 volts, to drive the reaction wheel motor. With an input range of ±25 WDE can be modelled as a simple amplifier with a gain of one. This stage also includes a 3.83 second lag compensation circuit that should be modelled if this is stage is to accurately represent the temporal operation of the WDE. Unfortunately, the orbital handbook only states that this lag filter exists, and does not give the *s*-domain transfer function.

```
(define-class WCE
              (instvars
                (gain  -17.75
                (Xn    0)
                (Xn-1  0)
                (Yn    0)
                (Yn-1  0)
                (Yn-2  0)
                (limit  25)
                (tolerance 0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))
(compile-class WCE)

(define-method (WCE amp-state)
  ()
  (let ( (Xn (send (eval (car input-list)) get-state)) )
    (if (null? Xn)
        nil
        (let* ( (Yn (+ (* 2.56 Xn) (* -2.4 Xn-1)
                       (* 1.2 Yn-1) (* -0.36 Yn-2) ))
                (vo (* gain Yn))
                (upper-rail limit)
                (lower-rail (* limit -1)) )
          (set! Yn-2 Yn-1)
          (set! Yn-1 Yn)
          (set! Xn-1 Xn)
          (cond ( (or (in-range? state vo tolerance)
                      (and (>? vo upper-rail)
                           (equal? state upper-rail) )
                      (and (<? vo lower-rail)
                           (equal? state lower-rail) ))
                  nil )
                (else
                  (cond
                    ( (>? vo upper-rail)
                      upper-rail )
                    ( (<? vo lower-rail)
                      lower-rail )
                    (else
                      vo ))))))))
```

Listing 11 - WCE Class Definition

Fortunately, the basis of a difference equation for a 3.83 second lag filter can be derived from the general transfer function of a lag-compensating filter, but with an unknown variable, α, to be determined.

The general lag filter transfer function is given by

$$H(s) = \frac{Ts + 1}{\alpha Ts + 1} \quad .$$

For $T = 3.83$ seconds, this becomes

$$H(s) = \frac{3.83s + 1}{3.83\alpha s + 1} \quad .$$

To derive the difference equation, we first transform the transfer function to the *z*-domain by substituting $1 - z^{-1}$ for *s*. The transfer function becomes

$$H(z) = \frac{Y(z)}{X(z)} = \frac{3.83(1-z^{-1}) + 1}{3.83\alpha(1-z^{-1}) + 1} = \frac{4.83 - 3.83z^{-1}}{(3.83\alpha + 1) - 3.83\alpha z^{-1}} \quad .$$

As before, cross multiplying and substituting the difference-equation relations for the *z* values, and putting the result in terms of the output y(n), gives

$$y(n) = \frac{4.83x(n) - 3.83x(n-1) + 3.83\alpha y(n-1)}{3.83\alpha + 1} \quad .$$

This difference equation will be used to model the 3.83 lag filter in the WCE. Using a Quattro™ simulation of the WCE compensation filter and the WDE lag filter in cascade, an α value of 4.5 was found to provide the best overall response to a unit step input. The step response of both filters individually, and together in cascade are shown in Figure 22. The WDE class definition is shown in Listing 12.

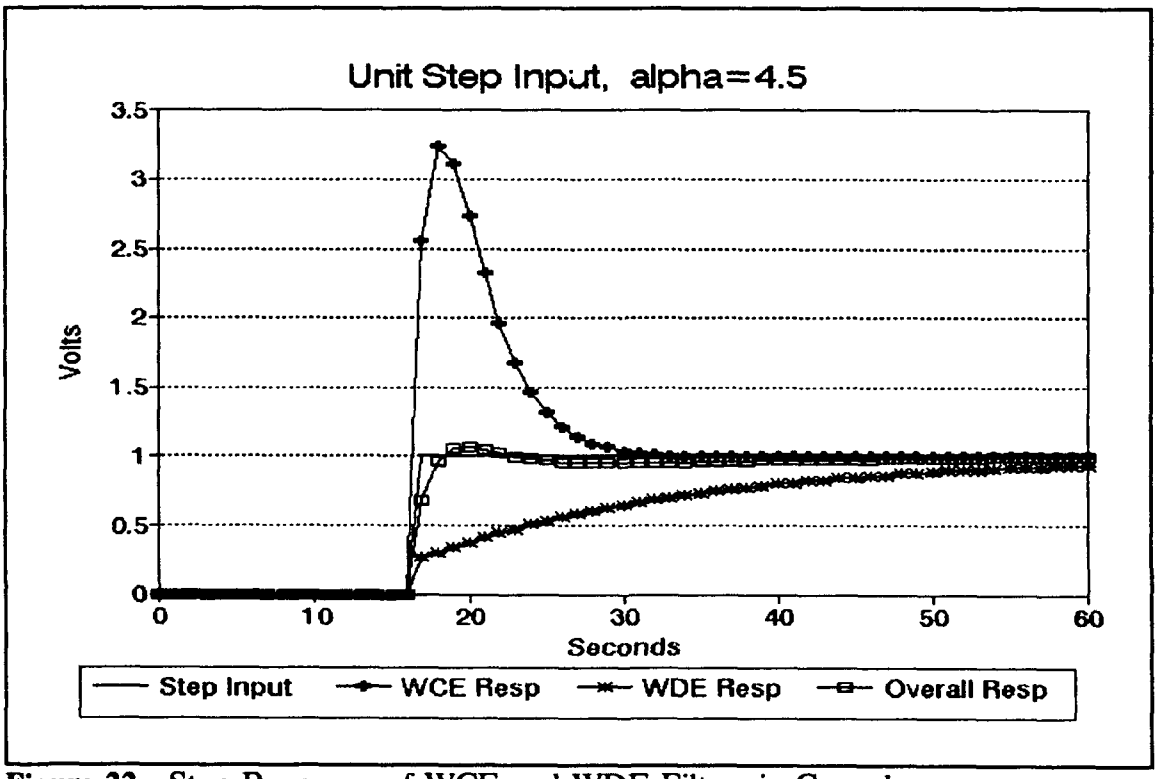

Figure 22 - Step Response of WCE and WDE Filters in Cascade

```
(define-class WDE
              (instvars
                (gain  1)
                (limit  25)
                (Xn     0)
                (Xn-1   0)
                (Yn     0)
                (Yn-1   0)
                (tolerance 0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))
(compile-class WDE)

(define-method (WDE amp-state)
  ()
  (let ( (Xn (send (eval (car input-list)) get-state)) )
    (if (null? Xn)
        nil
        (let* ( (Yn (/ (+ (* 4.83 Xn) (* -3.83 Xn-1)
                          (* 3.83 *ALPHA* Yn-1) )
                       (+ (* 3.83 *ALPHA*) 1) ))
                (vo (* gain Yn))
                (upper-rail limit)
                (lower-rail (* limit -1)) )
          (set! Yn-1 Yn)
          (set! Xn-1 Xn)
          (cond ( (or (in-range? state vo tolerance)
                      (and (>? vo upper-rail)
                           (equal? state upper-rail) )
                      (and (<? vo lower-rail)
                           (equal? state lower-rail) ))
                  nil )
                (else
                  (cond
                    ( (>? vo upper-rail)
                      upper-rail )
                    ( (<? vo lower-rail)
                      lower-rail )
                    (else
                      vo ))))))))
```

Listing 12 - WDE Class Definition

Reaction Wheel Assembly

The input voltage is applied to the wheel motor to create a torque that accelerates or decelerates the wheel. This torque is applied equally, in opposite directions, to the wheel's moment of inertia (I_w) and the spacecraft's moment of inertia (I_s). An object's *moment of inertia*, moving in a rotational frame, is the analog of an object's *mass* moving in a linear frame. Therefore, equations used to calculate linear quantities can be used to determine the angular equivalents, by replacing the linear quantities with the angular quantities shown in Table 4.

Table 4
Analogous Linear and Angular Quantities

Linear		Angular	
Displacement	s	Displacement	Θ
Velocity	v	Velocity	ω
Acceleration	a	Acceleration	α
Mass (inertia)	m	Moment of Inertia	I
Force	F = ma	Torque	$\tau = I\alpha$
Momentum	mv	Momentum	$I\omega$

On the spacecraft, a sensor provides a differential wheel speed. This deviation from the nominal 3000 rpm can be generated by the model, but the value alone cannot be used to determine how the acceleration of the wheel causes a rotation of the spacecraft about the pitch axis. The amount of physical rotation of the spacecraft is required to close the loop. This rotation reduces the error detected and output by the ESA. The data in Table 2 and Table 3, in the last chapter, provide enough information to derive equations that can help determine the angular displacement of the spacecraft given a reaction wheel speed change.

What is needed is a value for angular acceleration of the wheel (α_w), for a given voltage input. The angular acceleration can be used to determine the angular velocity changes in the wheel and hence the differential wheel speed. Also, since the torque on the wheel is equal to the torque on the spacecraft, but in the opposite direction, the angular acceleration of the wheel can be used to determine the angular acceleration of the spacecraft, provided the ratio of the moment of inertia of the wheel to that of the spacecraft is known.

The data from Table 3, show that the maximum input voltage coming from the WDE is ±25 Volts. With 25 volts applied, the motor generates a torque, to accelerate the wheel, of 15 in-oz, or 0.106 N-m in MKS units. Angular momentum

of the wheel is given as 6.77 ft-lb-s, or 9.18 N-m-s. Since angular momentum is given as $I\omega$ and ω = 3000 rpm or 314.16 rads/sec, then the moment of inertia for the wheel, I_w = 0.02922 Kg-m^2. Using the torque equation, $\tau = I\alpha$, α_w can be determined from

$$\alpha_w = \frac{\tau}{I_w} = \frac{0.106(N-m)}{0.02922(Kg-m^2)} = 3.63 r/s^2 \quad .$$

Therefore, a voltage of 25 V gives an angular acceleration of 3.63 r/s^2, or more conveniently, 34.66 rpm/s. Assuming a linear relationship between input voltage and acceleration, this results in a transfer function of 1.386 rpm/s/v.

The change in differential speed can be calculated as

$$\Delta\omega = \omega_f - \omega_o = \alpha_w t = 3.63t \quad rads/s \quad ,$$

or, in terms of rpm, the new differential wheel speed can be calculated as

$$\Delta rpm_f = \Delta rpm_o + \alpha_w t \quad .$$

Again, since the sample time is one second, the t can be dropped from the equations.

The transfer function (gain) and the differential wheel speed (delta-rpm) are used in the WHEEL class definition shown in Listing 13. The negative sign on the transfer function ensures that the spacecraft will rotate in the right direction to null out the pitch error.

Earth Sensor Assembly

As discussed in the last chapter, the ESA outputs a pitch error angle determined from chord lengths of a scanning beam that picks up radiance from the earth. The ESA is not modelled here on the way it works internally. Instead, the computer model of the ESA block will output a pitch error angle determined from the rotation of the spacecraft. The angular displacement is calculated from the acceleration, α_w, of the reaction wheel.

```
(define-class wheel
              (instvars
                (gain          -1.386)
                (delta-rpm      0)
                (limit          35)
                (tolerance      0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))
(compile-class wheel)

(define-method (WHEEL amp-state)
  ()
  (let* ( (vin (send (eval (car input-list)) get-state)) )
    (if (null? vin)
        nil
        (let ( (alpha-wheel (* gain vin))
               (upper-rail limit)
               (lower-rail (* limit -1)) )
          (cond ( (or (in-range? state alpha-wheel tolerance)
                      (and (>? alpha-wheel upper-rail)
                           (equal? state upper-rail) )
                      (and (<? alpha-wheel lower-rail)
                           (equal? state lower-rail) ))
                  nil )
                (else
                  (cond
                    ( (>? alpha-wheel upper-rail)
                      (set! delta-rpm (+ delta-rpm upper-rail))
                      upper-rail )
                    ( (<? alpha-wheel lower-rail)
                      (set! delta-rpm (+ delta-rpm lower-rail))
                      lower-rail )
                    (else
                      (set! delta-rpm (+ delta-rpm alpha-wheel))
                      alpha-wheel ))))))))
```

Listing 13 - Wheel Class Definition

The pitch axis moment of inertia, I_s, of the spacecraft is given in Table 2 (page 57) as 640 slug-ft^2, or 867.5 km-m^2 in MKS units. By Newton's third law the torque on the spacecraft is equal and opposite to the torque on the wheel. Therefore,

$$\tau = \alpha_s I_s = \alpha_w I_w \quad .$$

Rearranging in terms of α_s, and substituting the values for moment of inertia gives

$$\alpha_s = \alpha_w \frac{I_w}{I_s} = \alpha_w(rpm/s)\frac{0.02922(kg-m^2)}{867.5(kg-m^2)}$$

$$= \frac{\alpha_w(rpm/s)}{29638.6} \quad rpm/s$$

$$= \frac{\alpha_w(rpm/s)}{283505} \quad rads/s^2 \quad .$$

This angular acceleration (in rads/s^2) is derived directly from the wheel acceleration (in rpm/s) with the transfer function (or gain term) of 1/283505. This acceleration causes the spacecraft to rotate about the pitch axis to reduce the displacement error Θ. The pitch control loop causes α_w to become smaller as Θ decreases.

The amount of angular displacement is given by

$$\theta = \omega_o t + \frac{1}{2}\alpha_s t^2 \quad rads \; .$$

For t = 1 second

$$\theta = \omega_o + \frac{\alpha_s}{2} \quad rads \; ,$$

and the new pitch error becomes

$$\theta_{new} = \theta_{old} - \theta = \theta_{old} - (\omega_o + \frac{\alpha_s}{2}) \quad .$$

Angular velocity, ω_o, and acceleration, α_w, are in radian units, but the pitch error angle, Θ_{OLD}, is stored in the ESA's output state as degrees, so an adjustment to the equation is made as

$$\theta_{new} = \theta_{old} - \theta = \theta_{old} - (\omega_o + \frac{\alpha_s}{2})\frac{180}{\pi} \quad deg \quad .$$

The above equations use angular velocity, ω. Again, letting sample time, t = 1 second, ω is calculated using

$$\omega_f = \omega_o + \alpha_w \quad rads/s \quad .$$

These equations are used in the method belonging to the ESA class to calculate the new pitch error angle. The ESA class definition and method are shown in Listing 14.

```
(define-class ESA
              (instvars
                (gain      (/ 1 283505))
                (omega      0)
                (limit      5) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class ESA)

(define-method (ESA amp-state)
  ()
  (let* ( (alpha-wheel (send (eval (car input-list)) get-state)) )
    (if (null? alpha-wheel)
        nil
        (let* ( (alpha-sat (* gain alpha-wheel))
                (omega-new (+ omega alpha-sat))
                (theta (- state (* (+ omega (/ alpha-sat 2)) 5.73)))
                (upper-rail limit)
                (lower-rail (* limit -1)) )
          (set! omega omega-new)
          (cond ( (and (in-range? 0 theta tolerance)
                       (in-range? 0 omega (/ tolerance 100)) )
                  nil )
                (else
                  (cond
                    ( (>? theta upper-rail)
                      upper-rail )
                    ( (<? theta lower-rail)
                      lower-rail )
                    (else
                      theta )))))))))
```

Listing 14 - ESA Class Definition and Method

Commands and Sensors

In the code, *command* objects are used to input some external value into an object in the model pitch control channel. They have no inputs, only an output state. Sensor objects are similar, but have no objects in their output list, but do take inputs. Sensor objects also maintain a list of upstream components. This list is the suspect-list following detection of a discrepancy in that sensor. The sensor-object's state should equal the state of the component connected to the input of the sensor. Since a sensor can break, it is modelled as an amplifier with a gain of one. Command and sensor object classes are defined in Listing 15.

```
(define-class COMMAND
              (mixins COMPONENT)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))
(compile-class COMMAND)

(define-class SENSOR
              (instvars
                upstream-components
                (gain  1)
                (limit 100)
                (tolerance  0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))
(compile-class SENSOR)
```

Listing 15 - Command and Sensor Class Definitions

Structure

The structure of the system is represented by the interconnection of the components in the pitch control channel. Components will be interconnected for two cases. The first case is for the structure that represents the computer *model.* It will have all its components prefixed with an "m". The second case is for the structure that represents the *real* pitch control channel. It will include primary and redundant components, post fixed with an "a" or "b" respectively, and all components prefixed with an "r". The structure is based on Figure 9 in the last chapter. A simplified diagram showing the model and real structure representations is illustrated in Figure 23.

Interconnection data can only be placed into instantiated components. That is, instances of the component classes defined above must first be declared before they can be connected. An example of instantiation and interconnection is shown in Listing 16. In SCOOPS's "make-instance" block, the connection of an object into the structure occurs when the object's input-list and output-list reflect what objects are connected to the input and output respectively. Listing 16 shows the first two

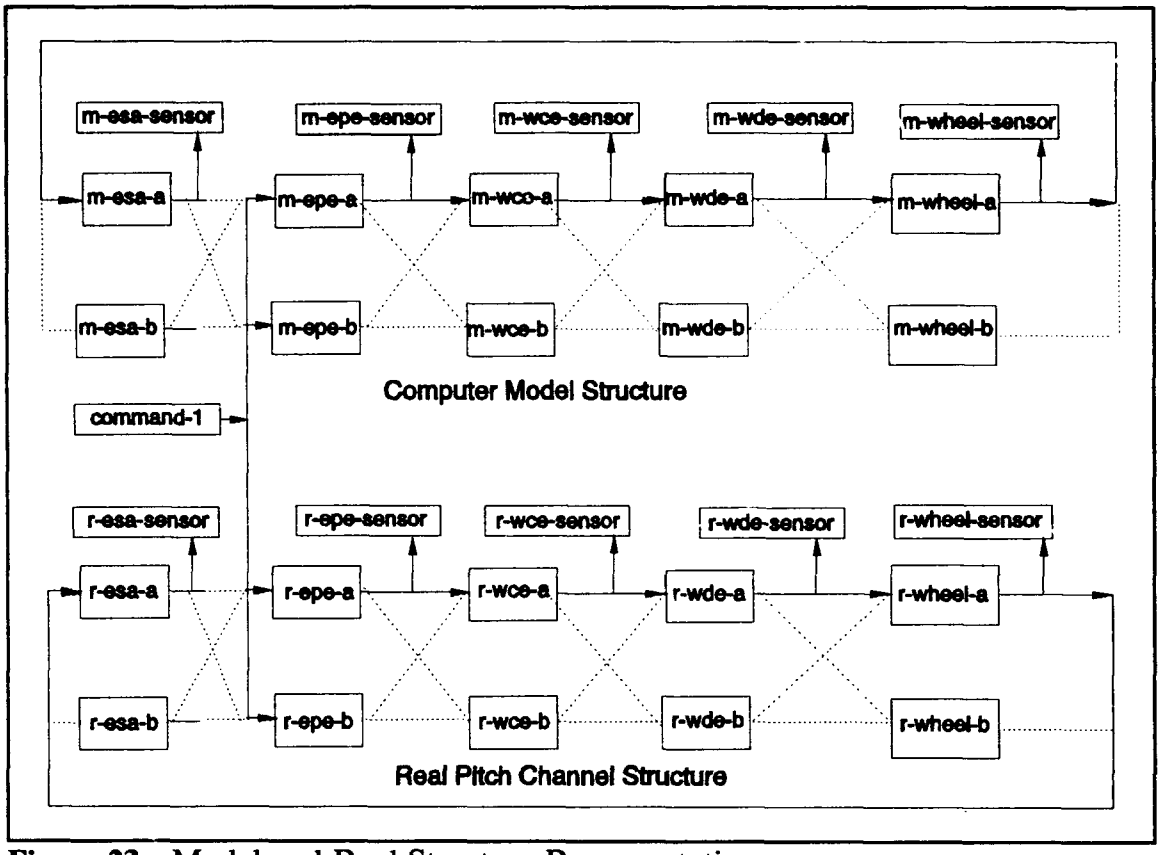

Figure 23 - Model and Real Structure Representations

components, the ESA and EPE, in the pitch control channel and the two sensors connected to their outputs. All objects are prefixed with an "m" so this example is a part of the computer ***m***odel. Note that in this case the pitch control loop has been broken between the ESA and the EPE, and a Command has been connected to the EPE input. The Command is connected to both the "real" and "model" structures, and will be used to enter a pitch error angle to test the model-based reasoner.

The upstream-components list in the sensor objects show what components upstream from that sensor are suspect if that sensor reports a discrepancy. In some model-based reasoning systems this list is built from a backward-search from the current sensor to a command input (Scarl, 1987:364). This search is important in a complex system that has many combinations of configuration states, but is difficult to do in a feedback system because mechanisms must be built into the software to stop

```
;;; earth sensor assembly
(define m-esa
  (make-instance ESA
      'name          'm-esa
      'output-list '(m-epe esa-sensor)
      'input-list  '(m-wheel) ))

;;; earth processing electronics
(define m-epe
  (make-instance EPE
      'name          'm-epe
      'output-list '(m-wce epe-sensor)
      'input-list  '(command-1) ))

;;; earth sensor assembly sensor
(define m-esa-sensor
  (make-instance sensor
      'name                  'm-esa-sensor
      'upstream-components  '(m-esa m-wheel m-wde m-wce m-epe)
      'input-list            '(m-esa) ))

;;; earth processing electronics sensor
(define m-epe-sensor
  (make-instance sensor
      'name                  'm-epe-sensor
      'upstream-components  '(m-epe)
      'input-list            '(m-epe) ))
```

Listing 16 - Example Instantiation and Interconnection of Components

the search. Instantiation of all objects showing the interconnection of the components in the model and the real pitch channel, is listed in files MPCHL.S and RPCHL.S, in Appendixes B and C respectively.

Another file that is associated with the structure is the PCONTROL.S file. This file, listed in Appendix D, includes functions and objects that are in common with, or somehow control, both the real and model pitch channels. For example, the *command-1* object in Figure 23 is instantiated in this file because its output is fed to both the model and real EPE. A function called *set-pitch* feeds an error signal into the circuit to initiate the simulation. This function also prints out the values of both the model and real sensors. Functions that change the configuration of the pitch control channel are also defined in this file. If the real pitch channel is reconfigured, the model is reconfigured accordingly. The model always reflects the current configuration of the real system. The PCONTROL.S file also maintains lists of all the components in the pitch channel. These lists are used by the configuration changer to replace objects in the input and output lists of each object to reflect

changes made to the configuration of the pitch control channel. Other miscellaneous "helper " functions are also kept in the PCONTROL.S file. One in particular, DESCRIBE-ALL, uses the SCOOPS built-in DESCRIBE function to print out the description of every instantiated object. This printout is listed in Appendix H, and shows the reset state of all objects in the real and model pitch-control channels.

Model-Based Reasoner

The algorithm used by the diagnostic reasoner in this thesis is based on Scarl's work, described in Chapter III on page 31. The assumptions and limitations Scarl makes for his *full consistency algorithm*, applied to LES, are relevant here. One limitation mentioned is that the algorithm does not work on feedback systems. Unfortunately, the pitch control system is one big feedback system; just as a technician breaks a feedback loop to diagnose a fault, the reasoner developed here will only be used when the loop is broken. The algorithm is described in Listing 17 and the detailed program listing is in file DIAGNOSE.S in the Appendix E. This

```
Algorithm:
     find a discrepant sensor
     if none found then
         no fault in circuit
     else
        collect all components structurally upstream from
           discrepant sensor and put into suspect list
        repeat for each suspect
           repeat for each fault hypothesis
              hypothesize a fault for the suspect
              propagate change through the model
              test all sensors for consistency
              if sensors consistent then
                 leave suspect in suspect list
              else
                 clear hypothetical fault (not suspect)
           end-repeat faults
           if all faults are ruled out then
                 clear suspect
        end-repeat suspects
        if one suspect remains then
           print out the culprit
        else
           print out the list of suspects remaining
```

Listing 17 - Reasoner Algorithm (based on Scarl's Full Consistency Algorithm)

algorithm has been described in detail in Chapter III and will not be discussed further

here. The *diagnose* function, that implements the algorithm, is written independently of any system. Provided the knowledge representation in the class definitions are adhered to, the *diagnose* function should operate on any system, within the assumptions made in Chapter III.

Two other functions, *fail* and *fix*, in the DIAGNOSE.S file, are not related to the diagnose algorithm, but provide a way of simulating a broken object. An object can be broken using the *fail* function. This function simply sets the object's *status* variable to one of the failure types listed in the object's *fault-list* variable. The *fix* function removes the fault condition from the object by resetting its *status* to "on".

Summary

This chapter has presented all aspects of software development using a model-based reasoning paradigm for fault detection in the pitch control channel of a satellite's attitude control subsystem. The chapter addressed language selection and why the Scheme language was chosen over Smalltalk, Lisp and C++. The chapter described the development of the objects used in the simulation. This was done by examination of the mathematical relationships between the object's input and output, and coming up with suitable equations to reflect the object's behavior. The chapter also described the interconnection of the objects that form the simulated pitch control channel. The pitch control channel is a small section of the larger attitude and velocity control subsystem. It was easily and quickly modelled using Scheme's object-oriented language. It should not be too difficult to apply this to the larger subsystem. Lastly, the chapter briefly touched on the algorithm used in the diagnosis process. The next chapter puts the software developed to the test.

VI. Results and Analysis

Introduction

This chapter will document the results and analysis of the software designed in the last chapter. That software uses a model-based reasoning paradigm to detect faults in a simulation of a pitch control channel of a typical geo-stationary satellite. Results and analysis will be presented in two steps. First, performance results of the designed computer model used to simulate the pitch control channel will be presented. Second, results and analysis of the performance of the model-based diagnostic reasoner will be presented.

Pitch Control Channel Simulation

The simulation of the pitch control channel was tested by closing the loop, and injecting a pitch error angle to initiate the simulation. The ESA sensor was monitored to determine how the system reacts to null out the error. The loop is not actually closed by simple connection of the ESA output to the EPE input. Instead, the loop is closed by inserting a function that takes the ESA output state and sends it to the EPE. The *test-loop* function, shown in Listing 18, carries out this task. It

```
(define (test-loop p)
  (reset-system)
  (set-tolerance .00000001)
  (send m-esa-a deposit-value p)
  (writeln)
  (writeln "m-esa-sensor" )
  (writeln (send m-esa-sensor get-state) )
  (run-test) )

(define (run-test)
  (send command-1 deposit-value (send m-esa-a get-state))
  (writeln (send m-esa-sensor get-state) )
  (run-test) )
```

Listing 18 - Function to Test Performance of the Pitch Control Channel Loop

takes an argument, the pitch error angle, and injects it into the ESA's *state* variable. Placing the *test-loop* function between the ESA and the EPE allows for observation and printout of the ESA-sensor data with each pass through the loop. It also prevents stack built-up and eventual memory-full errors that would occur if the system was connected in a loop and let the deposit-value function recursively propagate forever. The *reset-system* function sets the *state* of all objects to zero. This ensures that the same initial conditions exist for subsequent runs to enable comparisons to be made. The *set-tolerance* function sets the *tolerance* variable in all objects to a small value to ensure that the simulation does not stop. The esa-sensor values were sent to a file using Scheme's *transcript-on* function. This file was imported into Quattro™ and the results graphed.

Loop performance for an input step of 0.1 and 1.0 degree is shown in Figure 24. The "alpha" value, in the Figure, was discussed in the last chapter on page 87, and represents an unknown variable in the WDE compensation filter. In

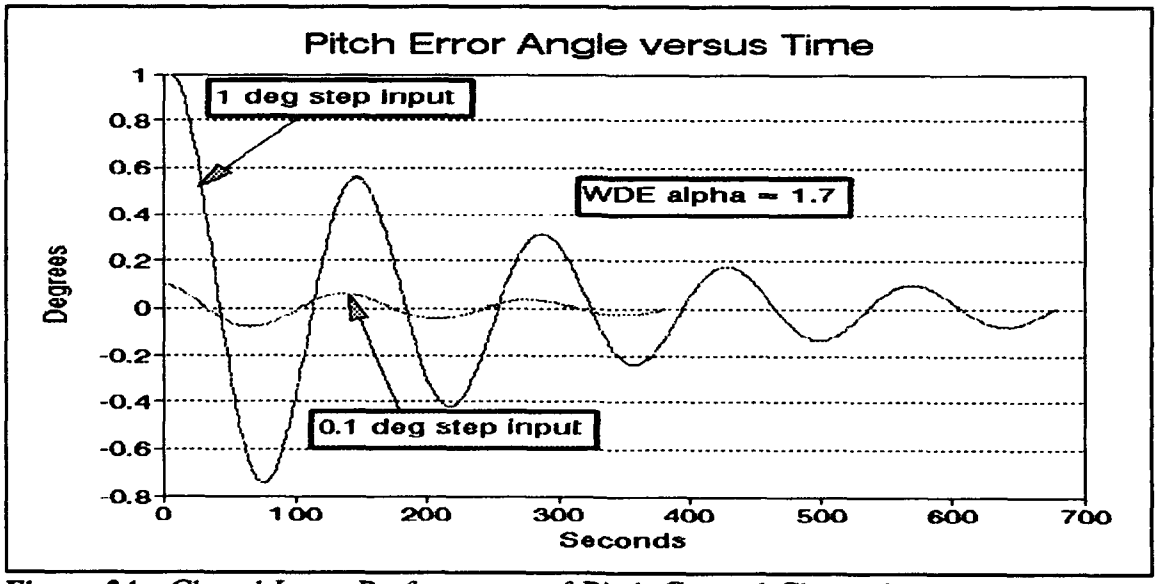

Figure 24 - Closed Loop Performance of Pitch Control Channel

that Chapter, a Quattro™ simulation of the difference equation for the lag filter was used to determine a value that would give the best response to a step input. The

value of 4.5 was chosen because it resulted in minimum overshoot and appeared to provide the best damping to the input step. When used in this closed loop simulation test, the system with alpha = 4.5 was found to be unstable! As shown in Figure 25, the pitch error signal grew larger instead of decaying as expected. The

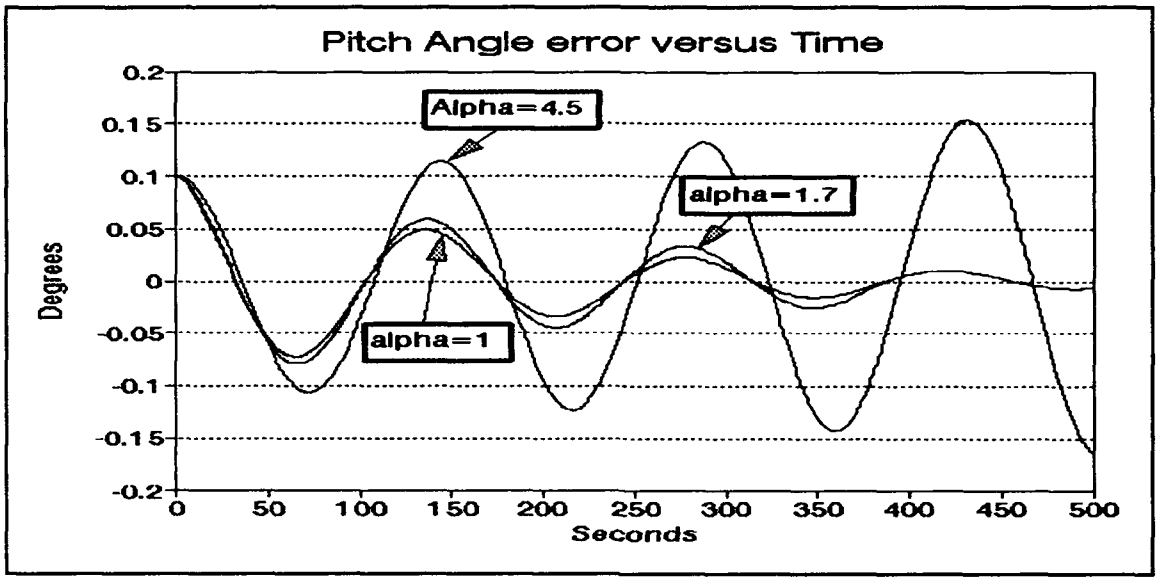

Figure 25 - Loop Performance for Three Alpha Values

reason for this is that the Quattro™ simulation in the last chapter was only checking the operation of the difference equations derived for the filters; it did not check the filters in the closed loop configuration with the wheel and spacecraft momentum taken into account. Two other values of alpha are shown in Figure 25: a value of 1.0 and a value of 1.7. Both these cases show the pitch error signal reducing with time, but the system is still underdamped. With reference to the lag filter transfer function equation in the last chapter, an alpha value of 1.0 reduces the filter to a simple buffer with a gain of one. That is, the filter is effectively out of circuit. This probably explains why so little emphasis was placed on it in the orbital operations handbook.

The orbital operations handbook specifies that pitch pointing accuracy be maintained within ±0.05 degrees. From the results of Figure 25, a step input of 0.1 degrees resulted in a settling time of about 150 seconds for both values of alpha (1.0

and 1.7). For an input step of 1.0 degree, Figure 24 shows the settling time to be about 700 seconds. This seems excessive and could be indicative of a problem with the computer model. But just how much rotational impulse does a spacecraft receive to shift it by as much as one degree? A clue to this comes from the discussion in Chapter III on page 57 on reaction wheel unloading. The reaction wheel's momentum is unloaded using a 0.05 second burst from a pitch thruster. This burst provides a 0.25 degree rotation about the pitch axis. There is a 1000 second delay before the next pitch firing to ensure the pitch error signal is fully decayed before another firing. This indicates that the settling times produced by this model is in the ball-park.

Now that the model has been checked serviceable, it is time to break it and see if the model-based reasoner can detect the fault.

Model-Based Reasoner Testing

Method of Operation

The following text describes the steps used to test the model-based reasoner. The first step is to set all objects to some initial state. This is done with the *reset-system* function. It simply reads in the MPCHL.S and RPCHL.S files and re-instantiates all the objects. Next, the *set-pitch* function injects an initial pitch error angle into the *r-esa-a*'s *state* variable and calls the *sp* function to propagate the signal through the rest of the system. The *sp* (for *s*et *p*itch) function takes the signal from the *r-esa-a* and uses *command-1* to inject it into the EPEs (model and real). The *sp* function is invoked by the user any time he wishes to propagate the pitch error signal from the ESA through to the rest of the real and model pitch control systems. In this way, the user can set any desired fault conditions using the *fail* function, run *sp* to propagate the ESA signal through the faulty system, and then invoke the *diagnose*

function to try and detect the fault in the system. The *sp* function also displays the value of the real and model world sensors with each invocation.

The above method of testing the system is done with the loop broken between the ESA and the EPE. Only the real world ESA is used in the control loop. Neither, the real nor model world ESAs are used in the diagnosis testing (Note the missing esa-sensor from the *all-sensor* list in the PCONTROL.S file) because they are used to model the spacecraft's reaction to the wheel's acceleration and do not reflect the actual operation of the real ESA. Furthermore, the real-world ESA's omega variable (instantaneous angular velocity of the spacecraft about the pitch axis), and model-world ESA's omega variable, quickly become different with diagnosis activity, Since the pitch error signal is calculated from this variable, the ESA-sensors would always be discrepant, and so not useful for diagnosis purposes. The *r-esa-a*'s pitch error signal is assumed to be good. This assumption should be valid one, because four pitch error signals are available from two on-board ESAs, and they could easily be corroborated.

Following a *set-pitch* or *sp* invocation, and with no fault conditions set, the real sensor values should equal the model sensor values. Invoking the *diagnose* function at this time should report "no fault found" because no discrepant sensors are detected. A fault condition is set using the *fail* function with an argument that represents the failure type. This must be followed with an *sp* invocation to propagate the signal through the faulty system. The model and real sensors should now show different values. Invoking the *diagnose* function now should detect the faulty component.

Following the detection of a faulty component, the system can be reconfigured to switch out the faulty component and switch in the redundant one. Another *sp* invocation is required to propagate the pitch signal through the newly configured

(hopefully fixed) system. Invoking *diagnose* again should confirm a fix by reporting "no fault found".

Results and Analysis - Designed Model

A sample Scheme program run that tested the reasoner against the pitch control channel models designed in Chapter V, is listed at Appendix I. With reference to that listing, the following is a description of its operation.

Lines 29 to 36 check the operation of the model-world and real-world pitch channel systems, and with no fault conditions set, there are no discrepant sensors. At line 37 a SCOOPS *describe* function was issued to look inside the *r-wce-a* object. The WCE was failed with a "latchup" failure mode at line 38 and confirmed with another look inside the WCE object at line 39. The latchup failure mode sets the output to +25 volts, and the WCE ignores any changes to the input. This was checked at line 40, and a discrepancy is confirmed between the real and model WCE sensors. This is shown in Listing 19. At line 41, a diagnose command is issued.

```
[40] (sp)

Model EPE-Sensor    = 0.472939960208774
Model WCE-Sensor    = -8.75638074294346
Model WDE-Sensor    = -9.45395266031472
Model Wheel-Sensor = 13.1031783871962
   wheel delta-rpm = 186.544048758552
Model ESA-Sensor    = 0.47756298633463
   esa omega        = 6.57992094525853e-4

Real EPE-Sensor    = 0.472939960208774
Real WCE-Sensor    = 25
Real WDE-Sensor    = 11.1190589701237
Real Wheel-Sensor = -15.4110157325915
  wheel delta-rpm = 131.603725387073
Real ESA Sensor    = 0.478652299563926
  esa omega        = 4.64202484566667e-4

DONE
```

Listing 19 - WCE Sensor Discrepancy

Debugging statements were left in the *diagnose* code to enable tracking of the reasoner algorithm. The result: the reasoner could not come up with the cause of the fault!

On closer examination, the reason for this became clear. The filters in the WCE and the WDE are memory devices. Their output is not only dependent on the current input, but also previous inputs and outputs. Recall, the difference equations used to model the loop compensation filter in the WCE is

$$y(n) = 2.56x(n) - 2.4x(n-1) + 1.2y(n-1) - 0.36y(n-2) \quad .$$

From this equation, it can be seen that more than one combination of input and output values (x(n), x(n-1)), y(n-1) and y(n-2)) can produce the same output y(n). This makes it difficult to determine the input, given an output, and hence almost impossible to find a fault model that could hypothesize the fault condition. This difficulty with devices that have memory has been mentioned often in the literature. In fact, in Chapter III, on this topic, Davis hinted that " ...future reasoning systems would have to go back through time from the original discrepancy, just like current reasoning systems go back through space [the circuit]."

Scarl also states in his paper (Scarl, 1987:362) on his LES implementation that "Feedback and objects whose outputs are not direct functions of their inputs (objects with state) have not been implemented, although doing so is high on our agenda." This thesis work confirms the difficulty researchers have had in implementing model-based reasoning on systems that have state that is dependent on time.

Results and Analysis - Modified Model

To enable successful diagnosis on the system means that either the reasoner algorithm has to change, or the model has to change. The former would be most desirable, but is beyond the scope of this thesis. Therefore, the model will be changed, but this change should be justified.

In Scarl's work on the LES prototype, one of his assumptions (Chapter III, page 35) was that his "full consistency algorithm" could handle systems where the

sensor polling cycle is shorter than the behavior changes in the system. The satellite pitch control channel is such a slow moving system. The closed-loop tests above show that the change in the pitch error signal is very small over a one second time interval. Consequent voltage changes through the rest of the system are also small over this time period. This means that a snapshot of the system state could be taken, and the diagnosis carried out over a one second period, without any serious loss of accuracy caused by slowly changing values.

The real and model-world pitch channels were changed by removing the filters in the WCE and the WDE, and treating them as simple amplifiers. This should approximate the pitch control channel for the short time between sensor polls. The structure of the new models is listed in files MPCHL2.S AND RPCHL2.S, in Appendixes F and G respectively. Because of the nature of object oriented programming, modifying the models was simple; the filters were removed by making the WCEs and WDEs instances of AMPLIFIER, with the gains set to the DC values calculated in Chapter V.

The closed-loop behavior of the system, without filters, was checked in the same manner as done for the system that included the filters (at the start of this chapter). It proved unstable! Instead of decaying, the pitch error signal oscillations grew with time. Various gain settings for the WCE did not help. The results are in Figure 26. The different gain values only changed the period of oscillation; the higher the gain, the shorter the period. A significant observation: the instability observed in the simulation without the filters, confirms that the difference equations did provide the loop compensation necessary to null out the pitch error.

The program run for this configuration is listed at Appendix J. A "verbose" version of a similar run, showing signal propagation detail, and the diagnostic mechanism is shown in Appendix K.

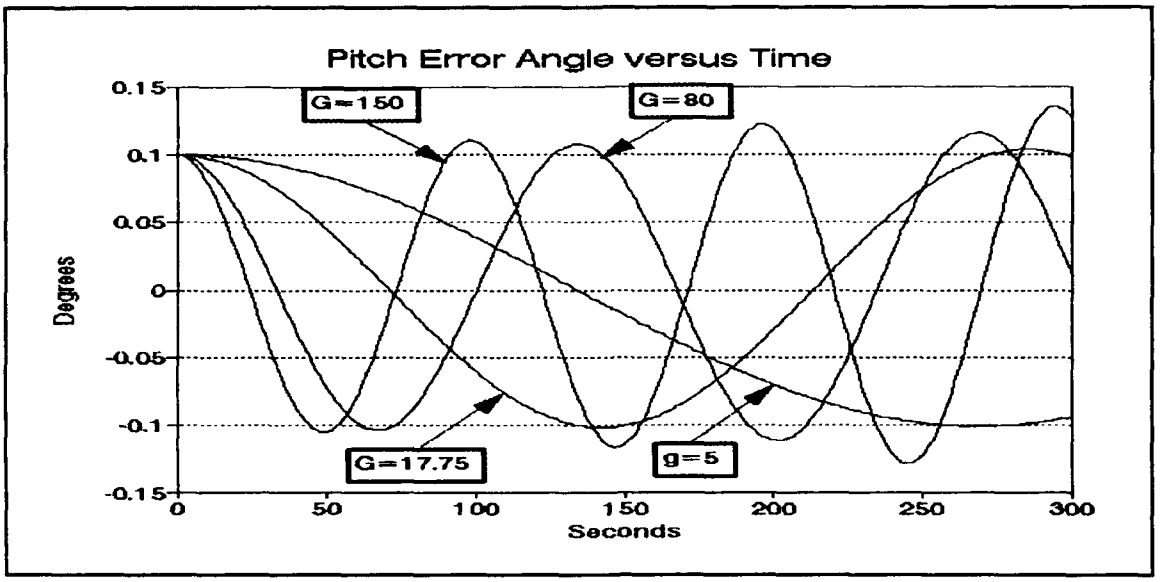

Figure 26 - Closed Loop Test - No Filters

The result: for almost all failure modes induced, the reasoner found the culprits. For one case it listed two suspects as the possible cause, and when two faults were induced, it detected the problem but could not come up with the culprits. With reference to Appendix J, the first *diagnose* command is issued at line 10, and the result returned is "*The culprit is: M-EPE-A*". The *diagnose* function uses the model-world pitch channel system to carry out its reasoning, hence reports the model-world EPE as being the culprit. This actually means that its the real-world equivalent, i.e., the *r-epe-a*, is cause of the current fault condition.

Of particular note in the results are:

- Reconfiguration commands ((epe-b-on) at line 11) successfully remove fault conditions (confirmation at line 13).
- Multiple faults could not be handled. This was expected since it was one of Scarl's assumptions. At line 28, the *r-wce-b* was failed, and was correctly diagnosed at line 30. Without reconfiguration or otherwise *fixing* this problem, the *r-wde-a* was failed at line 31. Diagnoses at line 33 detected a problem, but failed to find the cause.

- An unexpected result: Setting a fault condition into the last component in the system, *r-wheel-a* at line 38, resulted in a diagnosis of "The suspects are: (M-WHEEL-SENSOR M-WHEEL-A)" at line 40. On closer analysis, this is the correct result because the reasoner cannot eliminate the wheel's sensor, because there are no further sensors down the line to help eliminate it as a suspect.
- Sensor failures are detected as easily as fault conditions in other components. At line 44, the r-wce-sensor was failed. This is correctly diagnosed at line 46.

The above results are too good to be true. Are these valid result? The problem is that the reasoner is diagnosing faults in a system (the real-world simulation model) that is an exact copy of the computer model used by the reasoning process; right down to the last decimal place! This is too unrealistic. A better test would be to test the reasoner against a more realistic real-world simulation, but at the same time, use a computer model (for the reasoner) that does not include time dependent objects.

Results and Analysis - Combined Model

Testing the reasoner against a more realistic pitch channel simulation, yet not include any time dependent objects in the computer model, was achieved by using the designed real-world simulation model (i.e., with filters), and have the reasoner use the model-world pitch channel without filters. This was implemented easily by loading the model-world pitch channel (MPCHL2.S) on top of the already loaded designed-model. In this way the model-world objects that included the filters were re-instantiated to objects without filters.

The system was tested the same as before, with one exception. As before, the pitch error signal is propagated through the system with each invocation of the *sp* function. In this test, this function must be invoked several times to settle transients created whenever the system state is changed. Multiple invocations are necessary

because any abrupt change, such as the initial *set-pitch* function, or setting a failure mode in one of the objects, is like injecting a step function into the system that the filters cannot respond to instantaneously. This could be accommodated in an on-board diagnostic system by having it read sensor data, at ten second intervals, several times, before deciding that a sensor is discrepant. This would allow for the settling of any transients brought on by a sudden change in the system due to an abrupt pitch change (unlikely, unless the spacecraft was hit by a small meteorite), a malfunction, or even a configuration change (this would definitely cause a transient).

The test results for the reasoner using this combined model is listed at Appendix L. To save space, most of the sensor data generated by the *sp* function have been edited out. The results are almost identical to that of the previous test and show that the reasoner could detect and find the cause of all the faults induced. But the listing shown at Appendix L is the final run. Some earlier program runs were not so successful.

In the first program test run, the reasoner did not work at all. The problem was traced to the sensor discrepancy test mechanism, namely the *discrepant-sensor?* predicate in the DIAGNOSE.S file. It was comparing the real-world and model-world sensors against too narrow a range of values. The problem was solved by introducing a new variable for the sensor object called *discrep-range*, and setting its value to a higher value, hence a broader range. On the unsuccessful first test run, the *tolerance* variable in the sensor object was used as the range for testing sensor discrepancy. This narrow range resulted in many discrepant sensor readings, and the reasoner was attempting to diagnosed the system as though it had multiple faults. In the previous "designed model" and "modified model" test runs, the problem did not arise because the sensor values of the real-world sensors where identical to those of the model-world sensors.

The *discrep-range* value can be adjusted to give the desired results. If it is set too low, the range will be too narrow, and the system will detect a fault but will be unable to determine the cause; the same situation as just discussed. If it is set too high, the range will be too wide and it may not detect a fault at all. A value between one and five percent of the objects peak output swing (*limit* variable in the object) seems to provide an operable range for the *discrep-range* variable. It is a realistic range because tolerances given for typical electronic components fall in this range. For the sensors in the MPCHL2.S file (Appendix F), the *discrep-ranges* were set to 1 for the WCE, WDE, and WHEEL; and, .05 for the ESA and EPE.

With these *discrep-range* values set, the test run at Appendix L confirms that a model that contains no time dependant objects can be used to detect and isolate faults in a system that does contain time dependant objects. This is a significant result because it shows that the model used to diagnose faults in a real system does not necessarily have to be an accurate representation of that real system.

Another important observation from this test run is that during all the diagnosis activity, and even with the extra delay from the multiple *sp* invocations, the satellite pitch angle (from the ESA) changes very little. Since every *sp* invocation occurs once every second (imposed by the calculation of the pitch error angle in the ESA, and the derivation of the difference equations in the filters), each line number in the listing can represent a count in seconds. With reference to the listing, a fault condition is set at line 13, the fault is diagnosed at line 23, a redundant system switched in at line 24, and the system is back in working order at line 31. This sequence took 18 seconds. During this time, even with the system broken, the pitch error angle changed from 0.091 to 0.065 degrees. In this case, the change was probably small because the failure mode induced into *r-epe-a* was the "zero volt out" type, and so would not alter the speed of the reaction wheel. Another more extreme case, is an *r-wde-a* that has its output latched down (output set to -25 volts) at line

68. This would quickly drive the wheel to full deceleration. The fault is diagnosed at line 74 and the redundant system switched in at line 75. The system is back to normal at line 81. This took a total of 13 seconds, with the pitch error angle changing from 0.123 to 0.132 degrees during that time. These short times (compared to spacecraft's response to wheel speed changes) are significant because they include the extra time required to allow for settling of transients caused by sudden fault condition, and the switch-over to a redundant system.

The reasoner has been successful in detecting faults in the simulated pitch control channel. The simulation appears to accurately represent the real system in accordance with the specifications provided by the orbital operations handbook. However, the pitch control channel is a slow moving system. Faster systems could be accommodated by including the time variable, *t*, which was conveniently left out in the calculations using the angular motion equations (ESA class), and in the difference equations (WCE and WDE classes). In both cases it was set to one second. If the variable is included again, the time value could be set to some other, smaller value, to enable diagnosis and sensor sampling at a faster rate.

Although the reasoner was successful in this problem domain, it could do with some improvement. One area for improvement is the setting of fault conditions into the simulation model and the generation of fault hypotheses for the reasoner. A high (or low) fault condition was set into the simulation by simply adding (or subtracting) some fixed value to the object's output state. The reasoner hypothesized the fault condition by adding (or subtracting) the same fixed value. The reasoner could be improved by using symbolic values, such as "high", "low", and "ok", instead of (or as well as) the numeric values. These symbolic values could then be propagated through the system, and tested against the sensors of the real system. Of course, all methods would have to be modified to accommodate this symbolic propagation, as would the sensor discrepancy detection mechanism. Propagating symbolic data in this

way is called "qualitative reasoning", and several references to this is made in papers listed in the bibliography.

Summary

This chapter has documented the results and analyzed the performance of a model-based reasoning paradigm used to detect faults in the pitch control channel of the attitude and velocity control subsystem of a typical geo-stationary satellite. Results and performance analysis of a computer model used to simulate the pitch control channel was also presented. This model is a necessary part of the model-based reasoning paradigm, but was also used to model a real system whose components could be failed to enable testing of the diagnostic reasoner. The results were presented in two steps.

First, the computer model used to simulate the pitch control channel was tested. This closed loop test checked how an injected pitch error angle would propagate through the system, accelerate the reaction wheel, and rotate the spacecraft to null the error. This was a successful test as pitch decay time was comparable to that specified in the orbital operations handbook.

Second, the model-based reasoner itself was tested. Three separate tests were presented and analyzed. The first test using the designed model was unsuccessful. The reasoner could not work from a model whose objects have state that are dependent on time. The second test run was made using a modified model that replaced the time dependent filter objects with simple amplifiers. The reasoner worked perfectly with this model, but it was unrealistic to use such a model for the real pitch channel simulation. The third test used the designed model (with filters) for the real pitch channel simulation, and the modified model (without filters) for the reasoner to work with. This test was successful.

Two significant results came from the analysis. The first was that the model used to diagnose faults in a real system does not necessarily have to be an accurate representation of that real system. Second, the change in the pitch error signal, during the time required to detect and correct for a faulty condition, is quite small. This is important because an on-board diagnostic system of this type could quickly recover from a faulty condition in the pitch control channel, without the satellites communications system loosing a single bit of data due to antenna misalignment.

As with most software design projects, the process is usually iterative. This thesis work was no exception. During the final testing phase, changes to the software had to be quickly made to overcome problems encountered. This was made very easy because of the nature of object oriented programming. New objects were created effortlessly, as were the creation of new variables within objects. The Scheme language itself shares the credit for this ease in programming and resulted in very rapid prototyping.

Two improvements were suggested to enhance the operation of the reasoner. The first improvement was to include the time variable in the equations of angular motion, and in the difference equations. This would allow diagnosis of faster moving system, by sampling the sensors at a faster rate. The second improvement suggested the use of "qualitative" techniques to generate fault hypotheses. Symbolic values are propagated through the model instead of numeric values. This would enhance sensor discrepancy detection over a broader range of values.

A final note. Every effort has gone into making the computer simulation of the pitch control channel as realistic as possible, but it is still a computer simulation. This model-based reasoner, or any on-board diagnostic system should be validated using real hardware.

VII. Conclusions and Recommendations

Introduction

This chapter summarized the thesis work, draws conclusions from what has been found, and makes recommendations based on the analysis of the results.

Thesis Summary

Chapter II records a review of literature that looked at some current work using AI techniques in the diagnosis of satellite anomalies. The examples given were listed as prototypes, so it appears very few diagnostic systems are actually working in an operational environment. From the literature reviewed, it appears that the technique best suited for anomaly detection on an *autonomous* satellite is model-based reasoning. Compared to other systems (such as rule-based expert systems) that use knowledge about how a system fails, model-based reasoning uses knowledge about the way a system works. This enables model-based reasoning to detect faults not necessarily programmed into the knowledge base. On other AI systems, sensor validation became an overwhelming part of their knowledge base. In model-based reasoning, sensors are treated like any other components in the system, and need no special attention.

Chapter III records a review of model-based reasoning techniques presented by several authors. Davis's and Scarl's work was discussed in some detail. Davis's work was examined because it provided a good basis for the understanding of the model-based reasoning paradigm. Scarl's work was examined in detail because it was applied to a process control system that has many sensors, and had potential for use on a satellite's subsystem.

Chapter IV presented a part of a satellite subsystem that can be modelled for use in an automatic fault detection system employing model-based reasoning. The attitude velocity control subsystem (AVCS) of a typical geo-stationary satellite was selected to be the subsystem for experimentation. An overview of the function of the AVCS was given, but due to complexity, only the pitch channel of the AVCS was presented in enough detail to enable computer modelling for the model-based reasoner to operate on.

Chapter V presented all aspects of software development using a model-based reasoning paradigm for fault detection in the pitch control channel of a satellite's attitude control subsystem. The chapter addressed language selection and why the Scheme language was chosen over other languages. The chapter described the development of the objects used in the simulation, and the interconnection of the objects that form the simulated pitch control channel. The chapter also described the algorithm used in the diagnosis process.

Chapter VI documented the results and analysis of the designed model-based reasoning paradigm used to detect faults in the pitch control channel. The results were presented in two steps. The first presented results for the pitch control channel model, used by the reasoner, and as a simulation for a real system. The second, presented the results of the operation of the reasoner itself.

Conclusions

Conclusions drawn from this thesis work are listed as follows:

- Fully operational AI techniques used for the detection of anomalies in satellites are virtually non-existent. However, many prototype systems have been developed.

- The model-based reasoning paradigm promises a viable alternative to more traditional rule-based design for fault detection because it is capable of detecting faults that are not programmed into the knowledge base. This is particularly important for an autonomous satellite.
- Scarl's "full consistency algorithm" seems well suited for detection of faults in a satellite because the domain is similar to a process control system that has many sensors.
- The pitch control channel of the attitude and velocity control subsystem provides a good vehicle for testing the operation of a model-based reasoning fault detection system.
- The Scheme language, with its SCOOPS object oriented extension, provided fast prototyping of the pitch control channel model, and the reasoning software. The OOP structure of the language also enabled changes to be made very quickly to the software when problems were encountered.
- Based on data in the Orbital Operations Handbook, the model of the pitch control channel, provided a good simulation of the real system.
- Scarl's algorithm does not work with a model that uses feedback. The pitch control channel is a system with feedback, but the problem was eliminated by breaking the loop in the model used by the reasoner to detect the fault. The loop was not broken in the model used to simulate the real pitch control channel.
- Scarl's algorithm does not work with a model that uses objects whose state is dependent on time. This problem was overcome by modifying time dependent objects, in the model used by the reasoner, to be non-time dependent.
- The model used by the reasoner to diagnose faults does not necessarily have to be an accurate representation of the real system. In this thesis, the reasoner diagnosed from a model that had objects whose state *were not* dependent on time,

but successfully detected faults in a computer simulation model that used objects with state that *were* dependent on time. This should be extendible to application of the reasoner against a real hardware system that uses components, such as filters, that are time dependent.

- The change in the pitch error signal, during the time required to detect and correct for a faulty condition, is small. This is important because an on-board diagnostic system of this type could quickly recover from a faulty condition in the pitch control channel, without the satellites communications system loosing a single bit of data due to antenna misalignment.

Recommendations

Culminating from this thesis work, and from the conclusions listed above, the following recommendations are made.

- Validation of the software was carried out against a computer simulation of the pitch control channel because hardware and telemetry tapes were not available. For true validation, the software should be checked against hardware or at a very minimum, against an operating satellite using sensor data from telemetry tapes.
- Scarl's algorithm does not work with a model that has objects whose state is dependent on time. In this thesis, the problem was circumvented by modifying the time dependent objects to become non-time dependent. Fortunately, in this case, fault detection was still achievable, but this may not be the case for other, more complex, time dependent systems. Further research should be applied to algorithms that use the model-based paradigm with models that are time dependent.
- Another limitation with Scarl's algorithm, is that it does not handle feedback. Most satellite systems are control systems and are dependent on feedback for

their operation. In this thesis, the feedback problem was eliminated by breaking the loop, just as a technician does. This was successful for this relatively simple case, but other systems may require the loop to be closed, or may have loops within loops which must be modelled. Further research should be applied to models and algorithms that use the model-based paradigm to better handle feedback.

- The reasoner should be enhanced as suggested in the last chapter. The sample time variable should be included in difference equations and the equations of motion. This would allow diagnosis of faster moving systems. Qualitative techniques should be employed to generate fault hypotheses. This would enhance sensor discrepancy detection over a broader range of values.
- Fully autonomous satellite operation is in the distant future. This thesis has touched on a small part of making the transition to autonomy in the domain of satellite fault management. Work should continue in this area of automated fault detection and recovery, using the model-based reasoning paradigm. As this thesis has shown, this paradigm promises to be a very viable option.

Appendix A: Class Definition File Listing

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; File: PDEF.S
;;;
;;; AFIT/GSO/ENG/90D-03  THESIS
;;;
;;; MODEL BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES
;;;
;;; BY FLTLT RALPH W. DRIES
;;;
;;; DECEMBER 1990
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIPTION
;;;
;;; This file includes all the class definitions for objects
;;; in the pitch control channel.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; COMPONENT (class)
;;; This is the highest level object.  Other objects inherit instance
;;; variable defined here.  The STATE variable is only changed by the
;;  deposit-value method.  This method causes the change to recursively
;;  propogate through the system, starting with the connected components
;;; declared in the OUTPUT-LIST.

(define-class COMPONENT
              (instvars
                name
                (status 'on)
                (state  0)
                (input-list '())
                (output-list '()) )
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class COMPONENT)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;  DEPOSIT-VALUE (method of class COMPONENT)
;;  This method deposits a value as the state of the component and then
;;  updates the downstream-components for which the component is an input.
;;  This is done by calling the UPDATE method.

(define-method (COMPONENT deposit-value)
  (value)
;;;debug;;;(writeln "Change " name " to " value ".")
  (set! state value)
  (tell-other output-list) )

(define (tell-other output-list)
  (cond
    ( (null? output-list)
      nil )
```

```
    (else
      (send (eval (car output-list)) update)
      (tell-other (cdr output-list)) )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; AMPLIFIER (class)
;; This very simple amplifier has 1 inputs and 1 output, a gain, output
;; swing limits and a tolerance. that update checks against to prevent
;; endless looping in a feedback circuit.

(define-class AMPLIFIER
              (instvars
                gain
                limit
                (tolerance 0.01)
                (fault-list '(high low zero latchup latchdown)) )
              (mixins COMPONENT)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class AMPLIFIER)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; UPDATE (method for class AMPLIFIER)
;;;
;;; calls: amp-state
;;;
;;; called by: deposit-value

(define-method (AMPLIFIER update)
  ()
  (case status
    ( 'on         (let ( (vo (amp-state)) )
                    (if (number? vo)
                        (deposit-value vo)
                        nil )))
    ( 'off        (deposit-value nil) )
    ( 'latchup    (deposit-value limit) )
    ( 'latchdown (deposit-value (* limit -1)) )
    ( 'zero       (deposit-value 0) )
    ( 'high       (let ( (vo (amp-state)) )
                    (if (number? vo)
                        (deposit-value
                          (let ( (hi-val (+ vo 5)) )
                            (if (>? hi-val limit)
                                limit
                                hi-val )))
                        (deposit-value
                          (let ( (val (+ state 5)) )
                            (if (>? val limit)
                                limit
                                val ))))))
    ( 'low        (let ( (vo (amp-state)) )
                    (if (number? vo)
                        (deposit-value
                          (let ( (lo-limit (* limit -1))
                                 (lo-val (- vo 5)) )
                            (if (<? lo-val lo-limit)
                                lo-limit
                                lo-val )))
                        (deposit-value
                          (let ( (val (- state 5)) )
                            (if (<? val (* limit -1))
```

```
                                (* limit -1)
                                val ))))))
    (else (writeln "*** ERROR: Invalid status: " status)) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; AMP-STATE (method for class AMPLIFIER)
;;; Determines the output state from the inputs
;;;
;;; returns: new output state of AMPLIFIER
;;;
;;; calls: in-range?
;;;
;;; called by: update

(define-method (AMPLIFIER amp-state)
  ()
  (let ( (vi (send (eval (car input-list)) get-state)) )
    (if (null? vi)
        nil
        (let ( (vo (* gain vi))
               (upper-rail limit)
               (lower-rail (* limit -1)) )
          (cond ( (or (in-range? state vo tolerance)
                      (and (>? vo upper-rail)
                           (equal? state upper-rail) )
                      (and (<? vo lower-rail)
                           (equal? state lower-rail) ))
                  nil )
                (cond
                  ( (>? vo upper-rail)
                    upper-rail )
                  ( (<? vo lower-rail)
                    lower-rail )
                  (else
                    vo ))))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; IN-RANGE?  (general function)
;; is true if v2 falls in the range of v1.  The width of the
;; range is determined by the tolerance tol.
;; i.e.   v1 - tol < v2 < v1 + tol
;; v1 and v2 are first tested for null. If either is null, this proc returns
;; false to ensure the state is updated the first time through.

(define (in-range? v1 v2 tol)
  (if (or (null? v1) (null? v2))
      #F
      (let ( (upper (+ v1 tol))
             (lower (- v1 tol)) )
        (if (and (<? v2 upper) (>? v2 lower))
            #T
            #F ))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ESA (class)
;; This class defines the Earth Sensor Assembly.  In reality, the input is
;; the rotation of the spacecraft about the pitch (Y-axis).  In this case
;; the input comes from the wheel acceleration (alpha-wheel) which is in
;; the range of ±34.66 rpm/s.  Using the ratio of the moment of inertia
;; of the satellite to the moment of inertia of the wheel, the angular
;; acceleration (alpha-sat) of the satellite can be determined.  The gain is
;; the ratio of the moment of inertia of the wheel to that of the satellite.
;; The instantaneous angular velocity (omega) of the satellite is calculated
;; from the acceleration, and the change in pitch error angle is calculated
```

```
;; from the angular velocity.  The output state is the new pitch error angle.
;; The time unit used for the calculations is one second.  This assumes each
;; transition through the pitch control loop simulation takes one second.

(define-class ESA
              (instvars
                (gain     (/ 1 283505))
                (omega     0)
                (limit     5) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class ESA)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; AMP-STATE (method for class ESA)
;;; Determines the output state from the input.
;;;
;;; returns: new output state of ESA
;;;
;;; calls: in-range?
;;;
;;; called by: update

(define-method (ESA amp-state)
  ()
  (let* ( (alpha-wheel (send (eval (car input-list)) get-state)) )
    (if (null? alpha-wheel)
        nil
        (let* ( (alpha-sat (* gain alpha-wheel))
                (omega-new (+ omega alpha-sat))
                (theta (- state (* (+ omega (/ alpha-sat 2)) 5.73)))
                (upper-rail limit)
                (lower-rail (* limit -1)) )
          (set! omega omega-new)
;;;debug;;;           (writeln)
;;;debug;;;           (writeln "omega=" omega)
;;;debug;;;           (writeln  theta " " omega )
;;;debug;;;           (writeln "a-wheel=" alpha-wheel)
;;;debug;;;           (writeln "a-sat=" alpha-sat)
;;;debug;;;           (writeln)
          (cond ( (and (in-range? 0 theta tolerance)
                       (in-range? 0 omega (/ tolerance 100)) )
                  nil )
                (else
                  (cond
                    ( (>? theta upper-rail)
                      upper-rail )
                    ( (<? theta lower-rail)
                      lower-rail )
                    (else
                      theta ))))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EPE (class)
;; This is the Earth Processing Electronics class.  It is a decendant of
;; the AMPLIFIER, and inherits all methods and characteristics of the
;; the AMPLIFIER.  Essentially, the EPE is a digital to analog converter (DAC)
;; which converts the digital angle information from the earth sensor to
;; a voltage used be the following wheel control electronics.  The gain
;; here is the transfer function of the DAC and equals 0.976 volts/deg.
;; The output limits are ±5V.
```

```
(define-class EPE
              (instvars
                (gain  0.976)
                (limit  5)
                (tolerance 0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class EPE)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; WCE (class)
;; This is the class definition for the Wheel Control Electronics.  It is
;; a decendant of COMPONENT and inherits all its variables and methods.
;; The WCE is an amplifier and loop compensating filter that ensure correct
;; damping for the pitch channel control system.  The filter is modelled
;; using a difference equation.  It uses past history of the input and the
;; output.  Xn is the current input, Xn-1 is the previous input.  Yn is the
;; current output (determined from the difference eqn), Yn-1 the previous
;; output and Yn-2 the output before that.
;; The output limits are ±15V.

(define-class WCE
              (instvars
                (gain  -17.75)
                (Xn-1  0)
                (Yn-1  0)
                (Yn-2  0)
                (limit  25)
                (tolerance 0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class WCE)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; AMP-STATE (method for class WCE)
;;; Determines the output state from the inputs
;;;
;;; returns: new output state of WCE
;;;
;;; calls: in-range?
;;;
;;; called by: update

(define-method (WCE amp-state)
  ()
  (let ( (Xn (send (eval (car input-list)) get-state)) )
    (if (null? Xn)
        nil
        (let* ( (Yn (+ (* 2.56 Xn) (* -2.4 Xn-1) (* 1.2 Yn-1) (* -0.36 Yn-2)))
                (vo (* gain Yn))
                (upper-rail limit)
                (lower-rail (* limit -1)) )
          (set! Yn-2 Yn-1)
          (set! Yn-1 Yn)
          (set! Xn-1 Xn)
          (cond ( (or (in-range? state vo tolerance)
                      (and (>? vo upper-rail)
```

```
                        (equal? state upper-rail) )
                    (and (<? vo lower-rail)
                         (equal? state lower-rail) ))
                nil )
              (else
                (cond
                  ( (>? vo upper-rail)
                    upper-rail )
                  ( (<? vo lower-rail)
                    lower-rail )
                  (else
                    vo ))))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; WDE (class)
;; This is the Wheel Drive Electronics class.  It is a decendant of
;; AMPLIFIER, and inherits all methods and characteristics of AMPLIFIER.
;; The WDE is an amplifier that provides the correct drive
;; to accelerate or decelerate the reaction wheel, and to ensure correct
;; damping for the pitch channel control system.
;; Here, the WDE is modelled as a simple amplifier, with a gain determined
;; from DC transfer characteristics of the amplifier.
;; The output limits are ±25V.

(define-class WDE
              (instvars
                (gain  1)
                (limit  25)
                (Xn-1   0)
                (Yn-1   0)
                (tolerance 0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class WDE)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; AMP-STATE (method for class WDE)
;;; Determines the output state from the inputs
;;;
;;; returns: new output state of WDE
;;;
;;; calls: in-range?
;;;
;;; called by: update

(define-method (WDE amp-state)
  ()
  (let ( (Xn (send (eval (car input-list)) get-state)) )
    (if (null? Xn)
        nil
        (let* ( (Yn (/ (+ (* 4.83 Xn) (* -3.83 Xn-1) (* 3.83 *ALPHA* Yn-1))
                       (+ (* 3.83 *ALPHA*) 1) ))
                (vo (* gain Yn))
                (upper-rail limit)
                (lower-rail (* limit -1)) )
          (set! Yn-1 Yn)
          (set! Xn-1 Xn)
          (cond ( (or (in-range? state vo tolerance)
                      (and (>? vo upper-rail)
                           (equal? state upper-rail) )
                      (and (<? vo lower-rail)
```

```
                        (equal? state lower-rail) ))
                  nil )
                 (else
                   (cond
                     ( (>? vo upper-rail)
                       upper-rail )
                     ( (<? vo lower-rail)
                       lower-rail )
                     (else
                       vo ))))))))

;;; global variable *ALPHA* definition;;;;;;;;;;;;;;;;
(define *ALPHA* 1.7)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; WHEEL  (class)
;; This class defines the reaction wheel. tf is the transfer function.
;; Input Voltage range is ±25V and output change in rpm is 1.386 rpm/s/volt.
;; State is the acceleration rpm/s,  used to speed up or slow down the wheel.
;; This pitches the spacecraft.
;; Delta-rpm keeps the actual deviation of rpm from the 3000 rpm nominal.

(define-class wheel
              (instvars
                (gain        -1.386)
                (delta-rpm    0)
                (limit        35)
                (tolerance    0.0001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class wheel)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; WHEEL-STATE (method for class WHEEL)
;;; Determines the output state from the input
;;;
;;; returns: new output state of wheel
;;;
;;; calls: in-range?
;;;
;;; called by: update

(define-method (WHEEL amp-state)
  ()
  (let* ( (vin (send (eval (car input-list)) get-state)) )
    (if (null? vin)
        nil
        (let ( (alpha-wheel (* gain vin))
               (upper-rail limit)
               (lower-rail (* limit -1)) )
          (cond ( (or (in-range? state alpha-wheel tolerance)
                      (and (>? alpha-wheel upper-rail)
                           (equal? state upper-rail) )
                      (and (<? alpha-wheel lower-rail)
                           (equal? state lower-rail) ))
                  nil )
                (else
                  (cond
                    ( (>? alpha-wheel upper-rail)
                      (set! delta-rpm (+ delta-rpm upper-rail))
                      upper-rail )
```

```
                    ( (<? alpha-wheel lower-rail)
                      (set! delta-rpm (+ delta-rpm lower-rail))
                      lower-rail )
                    (else
                      (set! delta-rpm (+ delta-rpm alpha-wheel))
                      alpha-wheel )))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; SENSOR (class)
;; Each SENSOR has an input and no components connected.
;; The output state is equal to the input value.  A list of upstream
;; components is used as the set of suspects to check following
;; discrepant sensor detection.  This saves building a list of suspects
;; from the structure and functional discriptions in each object.  This
;; list must be changed following reconfiguration.  The limit value is
;; arbitrary, but could reflect a limit on a real on-board sensor.
;; The discrep-range value is used by the diagnoser to test
;; against a range when checking for discrepant sensors.  The default
;; tolerance is set to 0.5 volts

(define-class SENSOR
              (instvars
                upstream-components
                (gain  1)
                (limit 100)
                (discrep-range 0.5)
                (tolerance  0.001) )
              (mixins AMPLIFIER)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class SENSOR)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; COMMAND (class)
;; Each COMMAND has an output state only.  All instance variables
;; for this component are inherited from COMPONENT

(define-class COMMAND
              (mixins COMPONENT)
              (options
                gettable-variables
                settable-variables
                inittable-variables ))

(compile-class COMMAND)
```

Appendix B: Model Pitch Channel Listing

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; File: MPCHL.S
;;;
;;; AFIT/GSO/ENG/90D-03  THESIS
;;;
;;; MODEL BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES
;;;
;;; BY FLTLT RALPH W. DRIES
;;;
;;; DECEMBER 1990
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIPTION
;;;
;;; This file defines the structure, instances, and describes
;;; system interconnection for the Model pitch channel.
;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Primary A components

(define m-esa-a
  (make-instance ESA
      'name        'm-esa-a
      'output-list '(m-esa-sensor)
      'input-list  '(m-wheel-a) ))

(define m-epe-a
  (make-instance EPE
      'name        'm-epe-a
      'output-list '(m-wce-a m-epe-sensor)
      'input-list  '(command-1) ))

(define m-wce-a
  (make-instance WCE
      'name        'm-wce-a
      'output-list '(m-wde-a m-wce-sensor)
      'input-list  '(m-epe-a) ))

(define m-wde-a
  (make-instance WDE
      'name        'm-wde-a
      'output-list '(m-wheel-a m-wde-sensor)
      'input-list  '(m-wce-a) ))

(define m-wheel-a
  (make-instance WHEEL
      'name        'm-wheel-a
      'output-list '(m-esa-a m-wheel-sensor)
      'input-list  '(m-wde-a) ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Redundant B components
;;; On startup, all are powered off, (status = off), and inputs
;;; and outputs are connected to primary A components.

(define m-esa-b
  (make-instance ESA
      'name        'm-esa-b
      'status      'off
      'output-list '(m-esa-sensor)
      'input-list  '(m-wheel-a) ))

(define m-epe-b
  (make-instance EPE
      'name        'm-epe-b
      'status      'off
      'output-list '(m-wce-a m-epe-sensor)
      'input-list  '(command-1) ))

(define m-wce-b
  (make-instance WCE
      'name        'm-wce-b
      'status      'off
      'output-list '(m-wde-a m-wce-sensor)
      'input-list  '(m-epe-a) ))

(define m-wde-b
  (make-instance WDE
      'name        'm-wde-b
      'status      'off
      'output-list '(m-wheel-a m-wde-sensor)
      'input-list  '(m-wce-a) ))

(define m-wheel-b
  (make-instance WHEEL
      'name        'm-wheel-b
      'status      'off
      'output-list '(m-esa-a m-wheel-sensor)
      'input-list  '(m-wde-a) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SENSORS  (components, i.e., instance of SENSOR)
;;; The following defines the sensors connected to the output of
;;; gates in the circuit.

(define m-esa-sensor
  (make-instance SENSOR
      'name                'm-esa-sensor
      'upstream-components '(m-esa-sensor m-esa-a m-wheel-a m-wde-a m-wce-a
m-epe-a)
      'discrep-range       .05
      'input-list          '(m-esa-a) ))

(define m-epe-sensor
  (make-instance SENSOR
      'name                'm-epe-sensor
      'upstream-components '(m-epe-sensor m-epe-a)
      'discrep-range       .05
      'input-list          '(m-epe-a) ))

(define m-wce-sensor
  (make-instance SENSOR
      'name                'm-wce-sensor
      'upstream-components '(m-wce-sensor m-wce-a m-epe-a)
      'input-list          '(m-wce-a) ))
```

```
(define m-wde-sensor
  (make-instance SENSOR
      'name                'm-wde-sensor
      'upstream-components '(m-wde-sensor m-wde-a m-wce-a m-epe-a)
      'input-list          '(m-wde-a) ))

(define m-wheel-sensor
  (make-instance SENSOR
      'name                'm-wheel-sensor
      'upstream-components '(m-wheel-sensor m-wheel-a m-wde-a m-wce-a m-epe-a)
      'input-list          '(m-wheel-a) ))
```

Appendix C: Real Pitch Channel Listing

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; File: RPCHL.S
;;;
;;; AFIT/GSO/ENG/90D-03  THESIS
;;;
;;; MODEL BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES
;;;
;;; BY FLTLT RALPH W. DRIES
;;;
;;; DECEMBER 1990
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIPTION
;;;
;;; This file defines the structure, instances and describes the
;;; structure of the Real pitch channel.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Primary A components

(define r-esa-a
  (make-instance ESA
      'name        'r-esa-a
      'output-list '(r-esa-sensor)
      'input-list  '(r-wheel-a) ))

(define r-epe-a
  (make-instance EPE
      'name        'r-epe-a
      'output-list '(r-wce-a r-epe-sensor)
      'input-list  '(command-1) ))

(define r-wce-a
  (make-instance WCE
      'name        'r-wce-a
      'output-list '(r-wde-a r-wce-sensor)
      'input-list  '(r-epe-a) ))

(define r-wde-a
  (make-instance WDE
      'name        'r-wde-a
      'output-list '(r-wheel-a r-wde-sensor)
      'input-list  '(r-wce-a) ))

(define r-wheel-a
  (make-instance WHEEL
      'name        'r-wheel-a
      'output-list '(r-esa-a r-wheel-sensor)
      'input-list  '(r-wde-a) ))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Redundant B components
;;; On startup, all are powered off, (status = off), and inputs
;;; and outputs are connected to primary A components.
```

```
(define r-esa-b
  (make-instance ESA
      'name        'r-esa-b
      'status      'off
      'output-list '(r-esa-sensor)
      'input-list  '(r-wheel-a) ))

(define r-epe-b
  (make-instance EPE
      'name        'r-epe-b
      'status      'off
      'output-list '(r-wce-a r-epe-sensor)
      'input-list  '(command-1) ))

(define r-wce-b
  (make-instance WCE
      'name        'r-wce-b
      'status      'off
      'output-list '(r-wde-a r-wce-sensor)
      'input-list  '(r-epe-a) ))

(define r-wde-b
  (make-instance WDE
      'name        'r-wde-b
      'status      'off
      'output-list '(r-wheel-a r-wde-sensor)
      'input-list  '(r-wce-a) ))

(define r-wheel-b
  (make-instance WHEEL
      'name        'r-wheel-b
      'status      'off
      'output-list '(r-esa-a r-wheel-sensor)
      'input-list  '(r-wde-a) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SENSORS  (components, i.e., instance of SENSOR)
;;; The following defines the sensors connected to the output of
;;; gates in the circuit.

(define r-esa-sensor
  (make-instance SENSOR
      'name        'r-esa-sensor
      'input-list  '(r-esa-a) ))

(define r-epe-sensor
  (make-instance SENSOR
      'name        'r-epe-sensor
      'input-list  '(r-epe-a) ))

(define r-wce-sensor
  (make-instance SENSOR
      'name        'r-wce-sensor
      'input-list  '(r-wce-a) ))

(define r-wde-sensor
  (make-instance SENSOR
      'name        'r-wde-sensor
     'input-list  '(r-wde-a) ))

(define r-wheel-sensor
  (make-instance SENSOR
      'name        'r-wheel-sensor
      'input-list  '(r-wheel-a) ))
```

Appendix D: Pitch Control File Listing

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; File: PCONTROL.S
;;;
;;; AFIT/GSO/ENG/90D-03  THESIS
;;;
;;; MODEL BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES
;;;
;;; BY FLTLT RALPH W. DRIES
;;;
;;; DECEMBER 1990
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIPTION
;;;
;;; This file includes all the commands and functions that control
;;; both the real and model pitch control channels
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; COMMANDS  (components, i.e., instance of COMMAND)
;;; The following defines the command that initiates a state in
;;; components in the real and model worlds to start the simulation.

(define command-1
  (make-instance command
      'name                  'command-1
      'output-list '(m-epe-a m-epe-b r-epe-a r-epe-b) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SET-PITCH  (set-pitch p)
;;; This function sets a known pitch error signal, p, into the
;;; state of both model and real ESAs.
;;; This initiates a pitch error for the pitch control channel to
;;; operate on.  It calls the sp function to propogate the pitch
;;; error through the system.

(define (set-pitch p)
  (send m-esa-a set-state p)
  (send r-esa-a set-state p)
  (sp) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SP  (sp)
;;; This function takes the output of the r-ESA-a and put it into
;;; the EPEs via command-1.  When propogation of the signal ceases, all
;;; sensor data is printed out.

(define (sp)
  (send command-1 deposit-value (send r-esa-a get-state))
  (writeln)
  (writeln "Model EPE-Sensor   = " (send m-epe-sensor get-state))
  (writeln "Model WCE-Sensor   = " (send m-wce-sensor get-state))
  (writeln "Model WDE-Sensor   = " (send m-wde-sensor get-state))
```

```
  (writeln "Model Wheel-Sensor = " (send m-wheel-sensor get-state))
  (writeln "   wheel delta-rpm = " (send m-wheel-a get-delta-rpm))
  (writeln "Model ESA-Sensor   = " (send m-esa-sensor get-state))
  (writeln "   esa omega       = " (send m-esa-a get-omega))
  (writeln)
  (writeln "Real EPE-Sensor    = " (send r-epe-sensor get-state))
  (writeln "Real WCE-Sensor    = " (send r-wce-sensor get-state))
  (writeln "Real WDE-Sensor    = " (send r-wde-sensor get-state))
  (writeln "Real Wheel-Sensor = " (send r-wheel-sensor get-state))
  (writeln "  wheel delta-rpm = " (send r-wheel-a get-delta-rpm))
  (writeln "Real ESA Sensor    = " (send r-esa-sensor get-state))
  (writeln "  esa omega       = " (send r-esa-a get-omega))
  (writeln)
  'done )

(define (iter)
  (send command-1 deposit-value (send m-esa-a get-state)) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; GET-REAL-SENSOR (get-real-sensor sensor)
;;; This function returns the real world sensor equivalent of the
;;; model sensor.  If more sensors are added to the circuit,
;;; they must be added here.

(define (get-real-sensor sensor)
   (case sensor
      (m-epe-sensor r-epe-sensor)
      (m-wce-sensor r-wce-sensor)
      (m-wde-sensor r-wde-sensor)
      (m-wheel-sensor r-wheel-sensor)
      (else 'error) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ALL-SENSORS
;;; This function defines all the sensors required by the reasoner.
;;; The reasoner only works the the model sensors.  The addition
;;; of any more sensors to the system must be added here.

(define all-sensors
 '(m-epe-sensor m-wce-sensor m-wde-sensor m-wheel-sensor) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ALL-COMPONENTS
;;; This function defines all the components in the system.  The list is
;;; used by the reconfiguration commands to change attributes inside the
;;; objects when ever the system is reconfigured.  That is, components
;;; will have different components connected to their inputs and outputs.

(define all-real-components
  '(r-esa-a r-esa-b r-epe-a r-epe-b r-wce-a r-wce-b
    r-wde-a r-wde-b r-wheel-a r-wheel-b
    r-esa-sensor r-epe-sensor r-wce-sensor
    r-wde-sensor r-wheel-sensor ))

(define all-model-components
  '(m-esa-a m-esa-b m-epe-a m-epe-b m-wce-a m-wce-b
    m-wde-a m-wde-b m-wheel-a m-wheel-b
    m-esa-sensor m-epe-sensor m-wce-sensor
    m-wde-sensor m-wheel-sensor command-1 ))

(define all-components
  (append all-model-components all-real-components) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; POWER-ON configuration
```

```
;;; The following set of power-on commands, change the configuration of
;;; the pitch control channel.  For example, EPE-A-ON will power-on R-EPE-A,
;;; power-off R-EPE-B, change the input of the next stage (R-WCE-A and R-WCE-B)
;;; to take input from R-EPE-A, and change the output-list list to
;;; include R-EPE-A as connected.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; earth sensor assembly
(define (esa-a-on)
  (reconfigure all-real-components 'r-esa-b 'r-esa-a)
  (reconfigure all-model-components 'm-esa-b 'm-esa-a) )

(define (esa-b-on)
  (reconfigure all-real-components 'r-esa-a 'r-esa-b)
  (reconfigure all-model-components 'm-esa-a 'm-esa-b) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; earth processing electronics
(define (epe-a-on)
  (reconfigure all-real-components 'r-epe-b 'r-epe-a)
  (reconfigure all-model-components 'm-epe-b 'm-epe-a) )

(define (epe-b-on)
  (reconfigure all-real-components 'r-epe-a 'r-epe-b)
  (reconfigure all-model-components 'm-epe-a 'm-epe-b) )



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; wheel control electronics
(define (wce-a-on)
  (reconfigure all-real-components 'r-wce-b 'r-wce-a)
  (reconfigure all-model-components 'm-wce-b 'm-wce-a) )

(define (wce-b-on)
  (reconfigure all-real-components 'r-wce-a 'r-wce-b)
  (reconfigure all-model-components 'm-wce-a 'm-wce-b) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; wheel drive electronics
(define (wde-a-on)
  (reconfigure all-real-components 'r-wde-b 'r-wde-a)
  (reconfigure all-model-components 'm-wde-b 'm-wde-a) )

(define (wde-b-on)
  (reconfigure all-real-components 'r-wde-a 'r-wde-b)
  (reconfigure all-model-components 'm-wde-a 'm-wde-b) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; reaction wheel assembly
(define (wheel-a-on)
  (reconfigure all-real-components 'r-wheel-b 'r-wheel-a)
  (reconfigure all-model-components 'm-wheel-b 'm-wheel-a) )

(define (wheel-b-on)
  (reconfigure all-real-components 'r-wheel-a 'r-wheel-b)
  (reconfigure all-model-components 'm-wheel-a 'm-wheel-b) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; RECONFIGURE (reconfigure component-list from-item to-item)
;;; This function changes the configuration of the pitch control channel.
;;; The primary (A) and secondary (B) components are mutually exclusive.
;;; When one is switched on, the other is automatically switched off.
;;; This function goes through all the components in the pitch channel
;;; and adjusts the input and output lists of each component, to make sure
;;; it is connected to a switched "ON" component.
;;;
;;; called by: esa-a-on, esa-b-on, epe-a-on, epe-b-on,.... etc.
;;;
;;; calls: config
```

```
(define (reconfigure component-list from-item to-item)
  (send (eval to-item) set-status 'on)
  (send (eval from-item) set-status 'off)
  (config component-list from-item to-item) )

(define (config component-list from-item to-item)
  (cond ( (null? component-list)
          nil )
        (else
          (fixit (car component-list) from-item to-item)
          (config (cdr component-list) from-item to-item) )))

(define (fixit component from-item to-item)
  (let ( (i-lst (send (eval component) get-input-list))
         (o-lst (send (eval component) get-output-list)) )
       (if (null? i-lst)
           nil
           (send (eval component) set-input-list
             (replace from-item to-item i-lst) ))
       (if (null? o-lst)
           nil
           (send (eval component) set-output-list
             (replace from-item to-item o-lst) )))
  (if (member component all-sensors)
      (send (eval component) set-upstream-components
        (replace from-item to-item
          (send (eval component) get-upstream-components) ))
      nil ))

(define (replace from-item to-item lst)
  (cond ( (null? lst)
          nil )
        ( (equal? (car lst) from-item)
          (cons to-item (cdr lst)) )
        (else
          (cons (car lst) (replace from-item to-item (cdr lst))) )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; RESET-SYSTEM (reset-system)
;;; This function resets the state variable of all objects to 0.  It also
;;; sets other variables, such as yn, yn-1 etc. to 0.  The easiest way to
;;; reset all the objects is to re-instantiate the objects by reloading
;;; the structure files: mpch1.s and rpch1.s.   mpch12.s and rpch12.s have
;;; their components modelled as simple amplifiers (no filters).
;;;
;;; calls: reset-state, reset-wce, reset-wde
;;;
;;; called by: user

(define (reset-system)
  (load "mpch1.s")
  (load "rpch1.s") )

(define (reset-system2)
  (load "mpch12.s")
  (load "rpch12.s") )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SET-TOLERANCE (set-tolerance tol)
;;; This function sets the tolerance in each object to tol.
;;;

(define (set-tolerance tol)
  (set-tol all-components tol) )
```

```
(define (set-tol lst tol)
  (if (null? lst)
      nil
      (cond ( (equal? (car lst) 'command-1)
              (set-tol (cdr lst) tol) )
            (else
              (send (eval (car lst)) set-tolerance tol)
              (set-tol (cdr lst) tol) ))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; TEST-LOOP (tesp-loop p)
;;; This function effectively closes the pitch control channel loop
;;; so that loop behavior can be examined.
;;; The system is reset to initial conditions, and the tolerance set
;;; to a very small value to ensure all objects continue to pass on their
;;; state.
;;; Once setup, the run-test function is called to loop by passing esa
;;; state value on to the epe via command-1.

(define (test-loop p)
  (reset-system2)
  (set-tolerance .00000001)
  (send m-esa-a deposit-value p)
  (writeln)
  (writeln (send m-esa-sensor get-state) )
  (run-test) )

(define (run-test)
  (send command-1 deposit-value (send m-esa-a get-state))
  (writeln (send m-esa-sensor get-state) )
  (run-test) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIBE-ALL (describe-all lst)
;;; This function recursively calls the SCOOPS describe function to
;;; print out all instances of object in lst.

(define (describe-all lst)
  (cond ( (null? lst)
          nil )
        (else
          (describe (eval (car lst)))
          (describe-all (cdr lst)) )))
```

Appendix E: Model-Based Reasoner Listing

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; File: DIAGNOSE.S
;;;
;;; AFIT/GSO/ENG/90D-03  THESIS
;;;
;;; MODEL BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES
;;;
;;; BY FLTLT RALPH W. DRIES
;;;
;;; DECEMBER 1990
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIPTION
;;;
;;; This file defines the model-based reasoner.
;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DIAGNOSE
;;; This procedure does the diagnosis.
;;;
;;; The following algorithm is based on Scarl's Full Consistency Algorithm
;;;
;;; Algorithm:
;;;       find a discrepent sensor
;;;       if none found then
;;;           no fault in circuit
;;;       else
;;;          collect all components structurally upstream from
;;;             discrepent sensor and put into suspect list
;;;          repeat for each suspect
;;;             repeat for each fault hypothesis
;;;                hypothesize a fault for the suspect
;;;                propogate change through the model
;;;                test all sensors for consistency
;;;                if sensors consistent then
;;;                   leave suspect in suspect list
;;;                else
;;;                   clear hypothetical fault (not suspect)
;;;             end-repeat faults
;;;             if all faults are ruled out then
;;;                   clear suspect
;;;          end-repeat suspects
;;;          if one suspect remains then
;;;             print out the culprit
;;;          else
;;;             print out the list of suspects remaining

(define (diagnose)
  (let ( (bad-sensor (get-discrepant-sensor all-sensors)) )
;;;debug;;;(writeln "bad-sensor =" bad-sensor)
       (if (null? bad-sensor)
           (writeln "No fault found!")
```

```
          (let* ( (suspect-list (get-suspects bad-sensor) )
                  (final-suspects  (check-each suspect-list)) )
                (cond ( (null? final-suspects)
                        (writeln "*** Fault detected: Cause ~ ...own! ***") )

                      ( (equal? (length final-suspects) 1)
                        (writeln "The culprit is: " (car final-suspects)) )
                      (else
                        (writeln "The suspects are: " final-suspects) ))))))

(define (get-suspects sensor)
  (send (eval sensor) get-upstream-components) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CHECK-EACH  (check-each suspect-list)
;;; This procedure checks each suspect in the list to see if it
;;; is a possible cause.  It returns a list of remaining suspects.

(define (check-each suspect-list)
;;;debug;;;(writeln "suspect list =" suspect-list)
   (if (null? suspect-list)
       nil
       (if (check? (car suspect-list))
           (cons (car suspect-list) (check-each (cdr suspect-list)))
           (check-each (cdr suspect-list)) )))

;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CHECK? (check suspect)
;;; This procedure inverts the suspect's output, which
;;; automatically propogates through the system.  Downstream
;;; sensors are then checked for consistency.  The procedure
;;; inverts the suspect's output state back again before returning.
;;; It returns true if suspect is a probable cause, false otherwise.

(define (check? suspect)
  (let ( (fault-lst (send (eval suspect) get-fault-list))
         (current-status (send (eval suspect) get-status)) )
;;;debug;;;(writeln "checking suspect: " suspect)
       (cond ( (check-faults? suspect fault-lst)
               (send (eval suspect) set-status current-status)
               #T )
             (else
               (send (eval suspect) set-status current-status)
               #F ))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CHECK-FAULTS? (check-faults suspect fault-list)
;;; This procedure inverts the component's (item) output state.
;;; Using update ensures change of state will be automatically
;;; propogated to following components.

(define (check-faults? suspect fault-list)
  (if (null? fault-list)
      nil
      (if (check-hypoth? (car fault-list) suspect)
          #T
          (check-faults? suspect (cdr fault-list)) )))

(define (check-hypoth? fault-type suspect)
;;;debug;;;(writeln "checking fault: " fault-type)
  (send (eval suspect) set-status fault-type)
  (send (eval suspect) update)
  (let ( (ds (get-discrepant-sensor all-sensors)) )
;;;debug;;;    (writeln "discr sensor from all= " ds)
    (if (null? (get-discrepant-sensor all-sensors))
```

```
        #T
        #F )) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; GET-DISCREPANT-SENSOR (get-discrepant-sensor sensor-list)
;;; This procedure gets a discrepant sensor (if one exists) by comparing
;;; each sensor (in the sensor-list) in the real world against the one
;;; in the model world. The first discrepant sensor found is returned.
;;; If none are found '() is returned to indicate all is ok!
;;;  NOTE: Since a single point failure is assumed, any discrepant
;;;        sensor reading can be used to collect suspects.

(define (get-discrepant-sensor sensor-list)
  (cond
    ( (null? sensor-list)
      '() )
    ( (discrepant-sensor? (car sensor-list))
      (car sensor-list) )
    (else
        (get-discrepant-sensor (cdr sensor-list)) )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DISCREPANT-SENSOR? (discrepant-sensor? sensor)
;;; This procedure tests a sensor for discrepancy between the
;;; model-world sensors and the real-world sensors.  True is
;;; returned if there is a discrepancy, false otherwise.

(define (discrepant-sensor? sensor)
   (let* ( (r-sensor (get-real-sensor sensor))
           (r-value (send (eval r-sensor) get-state))
           (m-value (send (eval sensor) get-state))
           (d-range (send (eval sensor) get-discrep-range)) )
         (if (not (in-range? r-value m-value d-range))
            #t
            #f )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; FAIL  (fail component failure-type)
;;; This procedure fails a component in the real-world model simulation.
;;; The failure types allowed are listed in the object's fault-list.
;;; The output of the selected component to remain in the fault condition
;;; regardless of what the inputs are doing.
;;; eg. (fail r-epe-a 'latchup) the r prefix ensures only the real world
;;; epe is made to fail.

(define (fail component failure-type)
   (send component set-status failure-type) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; FIX  (fix component)
;;; This procedure fixes any component in the real-world objects that were
;;; broken using the fail function.
;;; eg. (fix r-epe-a)  will repair the epe object failed above.

(define (fix component)
   (send component set-status 'on) )
```

Appendix F: Model Pitch Channel Listing - No Filters

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; File: MPCHL2.S
;;;
;;; AFIT/GSO/ENG/90D-03  THESIS
;;;
;;; MODEL BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES
;;;
;;; BY FLTLT RALPH W. DRIES
;;;
;;; DECEMBER 1990
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIPTION
;;;
;;; This file defines the structure, instances and describes the
;;; structure of the modified Model pitch channel, with filters removed.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Primary A components

(define m-esa-a
  (make-instance ESA
      'name        'm-esa-a
      'output-list '(m-esa-sensor)
      'input-list  '(m-wheel-a) ))

(define m-epe-a
  (make-instance EPE
      'name        'm-epe-a
      'output-list '(m-wce-a m-epe-sensor)
      'input-list  '(command-1) ))

(define m-wce-a
  (make-instance AMPLIFIER
      'name        'm-wce-a
      'gain        -17.75
      'limit       25
      'output-list '(m-wde-a m-wce-sensor)
      'input-list  '(m-epe-a) ))

(define m-wde-a
  (make-instance AMPLIFIER
      'name        'm-wde-a
      'gain        1.0
      'limit       25
      'output-list '(m-wheel-a m-wde-sensor)
      'input-list  '(m-wce-a) ))

(define m-wheel-a
  (make-instance WHEEL
      'name        'm-wheel-a
      'output-list '(m-esa-a m-wheel-sensor)
      'input-list  '(m-wde-a) ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Redundant B components
;;; On startup, all are powered off, (status = off), and inputs
;;; and outputs are connected to primary A components.

(define m-esa-b
  (make-instance ESA
      'name        'm-esa-b
      'status      'off
      'output-list '(m-esa-sensor)
      'input-list  '(m-wheel-a) ))

(define m-epe-b
  (make-instance EPE
      'name        'm-epe-b
      'status      'off
      'output-list '(m-wce-a m-epe-sensor)
      'input-list  '(command-1) ))

(define m-wce-b
  (make-instance AMPLIFIER
      'name        'm-wce-b
      'status      'off
      'gain        -17.75
      'limit       25
      'output-list '(m-wde-a m-wce-sensor)
      'input-list  '(m-epe-a) ))

(define m-wde-b
  (make-instance AMPLIFIER
      'name        'm-wde-b
      'status      'off
      'gain        1.0
      'limit       25
      'output-list '(m-wheel-a m-wde-sensor)
      'input-list  '(m-wce-a) ))

(define m-wheel-b
  (make-instance WHEEL
      'name        'm-wheel-b
      'status      'off
      'output-list '(m-esa-a m-wheel-sensor)
      'input-list  '(m-wde-a) ))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SENSORS  (components, i.e., instance of SENSOR)
;;; The following defines the sensors connected to the output of
;;; gates in the circuit.

(define m-esa-sensor
  (make-instance SENSOR
      'name                'm-esa-sensor
      'upstream-components '(m-esa-sensor m-esa-a m-wheel-a m-wde-a m-wce-a
m-epe-a)
      'discrep-range       .05
      'input-list          '(m-esa-a) ))

(define m-epe-sensor
  (make-instance SENSOR
      'name                'm-epe-sensor
      'upstream-components '(m-epe-sensor m-epe-a)
      'discrep-range       .05
      'input-list          '(m-epe-a) ))
```

```
(define m-wce-sensor
  (make-instance SENSOR
      'name                  'm-wce-sensor
      'upstream-components '(m-wce-sensor m-wce-a m-epe-a)
      'input-list            '(m-wce-a) ))

(define m-wde-sensor
  (make-instance SENSOR
      'name                  'm-wde-sensor
      'upstream-components '(m-wde-sensor m-wde-a m-wce-a m-epe-a)
      'input-list            '(m-wde-a) ))

(define m-wheel-sensor
  (make-instance SENSOR
      'name                  'm-wheel-sensor
      'upstream-components '(m-wheel-sensor m-wheel-a m-wde-a m-wce-a m-epe-a)
      'input-list            '(m-wheel-a) ))
```

Appendix G: Real Pitch Channel Listing - No Filters

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; File: RPCHL2.S
;;;
;;; AFIT/GSO/ENG/90D-03  THESIS
;;;
;;; MODEL BASED REASONING IN THE DETECTION OF SATELLITE ANOMALIES
;;;
;;; BY FLTLT RALPH W. DRIES
;;;
;;; DECEMBER 1990
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DESCRIPTION
;;;
;;; This file defines the structure, instances and describes the
;;; structure of the modified Real pitch channel, with filters removed.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Primary A components

(define r-esa-a
  (make-instance ESA
      'name        'r-esa-a
      'output-list '(r-esa-sensor)
      'input-list  '(r-wheel-a) ))

(define r-epe-a
  (make-instance EPE
      'name        'r-epe-a
      'output-list '(r-wce-a r-epe-sensor)
      'input-list  '(command-1) ))

(define r-wce-a
  (make-instance AMPLIFIER
      'name        'r-wce-a
      'gain        -17.75
      'limit       25
      'output-list '(r-wde-a r-wce-sensor)
      'input-list  '(r-epe-a) ))

(define r-wde-a
  (make-instance AMPLIFIER
      'name        'r-wde-a
      'gain        1.0
      'limit       25
      'output-list '(r-wheel-a r-wde-sensor)
      'input-list  '(r-wce-a) ))

(define r-wheel-a
  (make-instance WHEEL
      'name        'r-wheel-a
      'output-list '(r-esa-a r-wheel-sensor)
      'input-list  '(r-wde-a) ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Redundant B components
;;; On startup, all are powered off, (status = off), and inputs
;;; and outputs are connected to primary A components.

(define r-esa-b
  (make-instance ESA
      'name        'r-esa-b
      'status      'off
      'output-list '(r-esa-sensor)
      'input-list  '(r-wheel-a) ))

(define r-epe-b
  (make-instance EPE
      'name        'r-epe-b
      'status      'off
      'output-list '(r-wce-a r-epe-sensor)
      'input-list  '(command-1) ))

(define r-wce-b
  (make-instance AMPLIFIER
      'name        'r-wce-b
      'status      'off
      'gain        -17.75
      'limit       25
      'output-list '(r-wde-a r-wce-sensor)
      'input-list  '(r-epe-a) ))

(define r-wde-b
  (make-instance AMPLIFIER
      'name        'r-wde-b
      'status      'off
      'gain        1.0
      'limit       25
      'output-list '(r-wheel-a r-wde-sensor)
      'input-list  '(r-wce-a) ))

(define r-wheel-b
  (make-instance WHEEL
      'name        'r-wheel-b
      'status      'off
      'output-list '(r-esa-a r-wheel-sensor)
      'input-list  '(r-wde-a) ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; SENSORS  (components, i.e., instance of SENSOR)
;;; The following defines the sensors connected to the output of
;;; gates in the circuit.

(define r-esa-sensor
  (make-instance SENSOR
      'name        'r-esa-sensor
      'input-list  '(r-esa-a) ))

(define r-epe-sensor
  (make-instance SENSOR
      'name        'r-epe-sensor
      'input-list  '(r-epe-a) ))

(define r-wce-sensor
  (make-instance SENSOR
      'name        'r-wce-sensor
      'input-list  '(r-wce-a) ))
```

```
(define r-wde-sensor
  (make-instance SENSOR
      'name         'r-wde-sensor
     'input-list   '(r-wde-a) ))

(define r-wheel-sensor
  (make-instance SENSOR
      'name         'r-wheel-sensor
      'input-list   '(r-wheel-a) ))
```

Appendix H: SCOOPS Component Descriptions

The following listing describes the structure of all objects instantiated in the pitch control channel. The listing was generated using the *describe-all* function with the parameter *all-components* passed to it. The structure, and reset-state of each object in the *all-components* list is displayed using SCOOPS inbuilt *describe* function. The first few statements, immediately below, reset the structure of the pitch control channel and set the tolerance in each object to 0.01.

```
[3] (reset-system)
OK
[4] (set-tolerance .01)
()
[5] (describe-all all-components)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class ESA

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   TOLERANCE : 0.01
   NAME : M-ESA-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-WHEEL-A)
   OUTPUT-LIST : (M-ESA-SENSOR)
   GAIN : 3.52727465124072e-6
   OMEGA : 0
   LIMIT : 5

  INSTANCE DESCRIPTION
  ====================

 Instance of Class ESA

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   TOLERANCE : 0.01
   NAME : M-ESA-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (M-WHEEL-A)
   OUTPUT-LIST : (M-ESA-SENSOR)
   GAIN : 3.52727465124072e-6
   OMEGA : 0
   LIMIT : 5

  INSTANCE DESCRIPTION
  ====================
```

```
 Instance of Class EPE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-EPE-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (M-WCE-A M-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class EPE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-EPE-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (M-WCE-A M-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WCE-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-EPE-A)
   OUTPUT-LIST : (M-WDE-A M-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0
   YN-1 : 0
   YN-2 : 0
   LIMIT : 25
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WCE-B
   STATUS : OFF
```

```
   STATE : 0
   INPUT-LIST : (M-EPE-A)
   OUTPUT-LIST : (M-WDE-A M-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0
   YN-1 : 0
   YN-2 : 0
   LIMIT : 25
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WDE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WDE-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-WCE-A)
   OUTPUT-LIST : (M-WHEEL-A M-WDE-SENSOR)
   GAIN : 1
   LIMIT : 25
   XN-1 : 0
   YN-1 : 0
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WDE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WDE-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (M-WCE-A)
   OUTPUT-LIST : (M-WHEEL-A M-WDE-SENSOR)
   GAIN : 1
   LIMIT : 25
   XN-1 : 0
   YN-1 : 0
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WHEEL

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WHEEL-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-WDE-A)
   OUTPUT-LIST : (M-ESA-A M-WHEEL-SENSOR)
   GAIN : -1.386
```

```
   DELTA-RPM : 0
   LIMIT : 35
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WHEEL

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WHEEL-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (M-WDE-A)
   OUTPUT-LIST : (M-ESA-A M-WHEEL-SENSOR)
   GAIN : -1.386
   DELTA-RPM : 0
   LIMIT : 35
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-ESA-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-ESA-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS  :  (M-ESA-SENSOR  M-ESA-A  M-WHEEL-A  M-WDE-A  M-WCE-A
M-EPE-A)
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.05
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-EPE-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-EPE-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : (M-EPE-SENSOR M-EPE-A)
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.05
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
```

```
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WCE-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-WCE-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : (M-WCE-SENSOR M-WCE-A M-EPE-A)
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WDE-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-WDE-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : (M-WDE-SENSOR M-WDE-A M-WCE-A M-EPE-A)
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : M-WHEEL-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (M-WHEEL-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : (M-WHEEL-SENSOR M-WHEEL-A M-WDE-A M-WCE-A M-EPE-A)
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class COMMAND

 Class Variables :
```

```
Instance Variables :
   OUTPUT-LIST : (M-EPE-A M-EPE-B R-EPE-A R-EPE-B)
   INPUT-LIST : ()
   STATE : 0
   STATUS : ON
   NAME : COMMAND-1

  INSTANCE DESCRIPTION
  ====================

 Instance of Class ESA

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   TOLERANCE : 0.01
   NAME : R-ESA-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-WHEEL-A)
   OUTPUT-LIST : (R-ESA-SENSOR)
   GAIN : 3.52727465124072e-6
   OMEGA : 0
   LIMIT : 5

  INSTANCE DESCRIPTION
  ====================

 Instance of Class ESA

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   TOLERANCE : 0.01
   NAME : R-ESA-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (R-WHEEL-A)
   OUTPUT-LIST : (R-ESA-SENSOR)
   GAIN : 3.52727465124072e-6
   OMEGA : 0
   LIMIT : 5

  INSTANCE DESCRIPTION
  ====================

 Instance of Class EPE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-EPE-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (R-WCE-A R-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
```

```
  ====================

 Instance of Class EPE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-EPE-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (R-WCE-A R-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-EPE-A)
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0
   YN-1 : 0
   YN-2 : 0
   LIMIT : 25
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (R-EPE-A)
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0
   YN-1 : 0
   YN-2 : 0
   LIMIT : 25
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WDE

 Class Variables :
```

```
Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WDE-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-WCE-A)
   OUTPUT-LIST : (R-WHEEL-A R-WDE-SENSOR)
   GAIN : 1
   LIMIT : 25
   XN-1 : 0
   YN-1 : 0
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WDE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WDE-B
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (R-WCE-A)
   OUTPUT-LIST : (R-WHEEL-A R-WDE-SENSOR)
   GAIN : 1
   LIMIT : 25
   XN-1 : 0
   YN-1 : 0
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WHEEL

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WHEEL-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-WDE-A)
   OUTPUT-LIST : (R-ESA-A R-WHEEL-SENSOR)
   GAIN : -1.386
   DELTA-RPM : 0
   LIMIT : 35
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WHEEL

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WHEEL-B
   STATUS : OFF
   STATE : 0
```

```
   INPUT-LIST : (R-WDE-A)
   OUTPUT-LIST : (R-ESA-A R-WHEEL-SENSOR)
   GAIN : -1.386
   DELTA-RPM : 0
   LIMIT : 35
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-ESA-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-ESA-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : #!UNASSIGNED
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-EPE-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-EPE-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : #!UNASSIGNED
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-WCE-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : #!UNASSIGNED
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
```

```
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WDE-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-WDE-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : #!UNASSIGNED
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
   TOLERANCE : 0.01

  INSTANCE DESCRIPTION
  ====================

 Instance of Class SENSOR

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WHEEL-SENSOR
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-WHEEL-A)
   OUTPUT-LIST : ()
   UPSTREAM-COMPONENTS : #!UNASSIGNED
   GAIN : 1
   LIMIT : 100
   DISCREP-RANGE : 0.5
   TOLERANCE : 0.01
()
[6] (transcript-off)
```

Appendix I: Program Run - Designed Model

The following Scheme program run starts is for the pitch channel simulation that includes the filters in the WCE and the WDE. The WCE was failed at line 38, the sensors checked at line 40 and a diagnosis carried out at line 41. The reasoner could not come up with the cause of the fault!

```
[28] (reset-system)
OK
[29] (set-tolerance .001)
()
[30] (set-pitch .5)

Model EPE-Sensor    = 0.488
Model WCE-Sensor    = -25
Model WDE-Sensor    = -15.0758734859112
Model Wheel-Sensor = 20.8951606514729
   wheel delta-rpm = 38.0367313297177
Model ESA-Sensor    = 0.499826773425537
   esa omega        = 1.34165998235367e-4

Real EPE-Sensor    = 0.488
Real WCE-Sensor    = -25
Real WDE-Sensor    = -15.0758734859112
Real Wheel-Sensor = 20.8951606514729
  wheel delta-rpm = 38.0367313297177
Real ESA Sensor    = 0.499826773425537
  esa omega        = 1.34165998235367e-4

DONE
[31] (sp)

Model EPE-Sensor    = 0.488
Model WCE-Sensor    = -25
Model WDE-Sensor    = -15.0758734859112
Model Wheel-Sensor = 20.8951606514729
   wheel delta-rpm = 38.0367313297177
Model ESA-Sensor    = 0.499826773425537
   esa omega        = 1.34165998235367e-4

Real EPE-Sensor    = 0.488
Real WCE-Sensor    = -25
Real WDE-Sensor    = -15.0758734859112
Real Wheel-Sensor = 20.8951606514729
  wheel delta-rpm = 38.0367313297177
Real ESA Sensor    = 0.499826773425537
  esa omega        = 1.34165998235367e-4

DONE
[32] (sp)

Model EPE-Sensor    = 0.488
Model WCE-Sensor    = -23.70477568
Model WDE-Sensor    = -15.4994523598706
Model Wheel-Sensor = 21.4822409707807
   wheel delta-rpm = 59.5189723004984
Model ESA-Sensor    = 0.4982832982623
```

```
   esa omega        = 2.09939762263446e-4

Real EPE-Sensor    = 0.488
Real WCE-Sensor    = -23.70477568
Real WDE-Sensor    = -15.4994523598706
Real Wheel-Sensor = 21.4822409707807
  wheel delta-rpm = 59.5189723004984
Real ESA Sensor    = 0.4982832982623
  esa omega        = 2.09939762263446e-4

DONE
[33] (set-tolerance .00001)
()
[34] (sp)

Model EPE-Sensor    = 0.486324499104005
Model WCE-Sensor    = -16.8910458677985
Model WDE-Sensor    = -13.2974221853134
Model Wheel-Sensor = 18.4302271488444
   wheel delta-rpm = 97.9088827718956
Model ESA-Sensor    = 0.495086023076669
   esa omega        = 3.45351520332607e-4

Real EPE-Sensor    = 0.486324499104005
Real WCE-Sensor    = -16.8910458677985
Real WDE-Sensor    = -13.2974221853134
Real Wheel-Sensor = 18.4302271488444
  wheel delta-rpm = 97.9088827718956
Real ESA Sensor    = 0.495086023076669
  esa omega        = 3.45351520332607e-4

DONE
[35] (sp)

Model EPE-Sensor    = 0.483203958522829
Model WCE-Sensor    = -12.4461685629118
Model WDE-Sensor    = -11.4678175841791
Model Wheel-Sensor = 15.8943951716722
   wheel delta-rpm = 130.802263494763
Model ESA-Sensor    = 0.490452314607523
   esa omega        = 4.61375508349986e-4

Real EPE-Sensor    = 0.483203958522829
Real WCE-Sensor    = -12.4461685629118
Real WDE-Sensor    = -11.4678175841791
Real Wheel-Sensor = 15.8943951716722
  wheel delta-rpm = 130.802263494763
Real ESA Sensor    = 0.490452314607523
  esa omega        = 4.61375508349986e-4

DONE
[36] (sp)

Model EPE-Sensor    = 0.478681459056942
Model WCE-Sensor    = -10.0258011315152
Model WDE-Sensor    = -10.2546791091417
Model Wheel-Sensor = 14.2129852452704
   wheel delta-rpm = 159.914140107163
Model ESA-Sensor    = 0.484569631361449
   esa omega        = 5.64061092774955e-4

Real EPE-Sensor    = 0.478681459056942
Real WCE-Sensor    = -10.0258011315152
Real WDE-Sensor    = -10.2546791091417
Real Wheel-Sensor = 14.2129852452704
```

```
  wheel delta-rpm = 159.914140107163
Real ESA Sensor   = 0.484569631361449
  esa omega       = 5.64061092774955e-4

DONE
[37] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-A
   STATUS : ON
   STATE : -10.0258011315152
   INPUT-LIST : (R-EPE-A)
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0.478681459056942
   YN-1 : 0.564833866564236
   YN-2 : 0.617228472790691
   LIMIT : 25
   TOLERANCE : 1.e-5
()
[38] (fail r-wce-a 'latchup)
LATCHUP
[39] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-A
   STATUS : LATCHUP
   STATE : -10.0258011315152
   INPUT-LIST : (R-EPE-A)
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0.478681459056942
   YN-1 : 0.564833866564236
   YN-2 : 0.617228472790691
   LIMIT : 25
   TOLERANCE : 1.e-5
()
[40] (sp)

Model EPE-Sensor   = 0.472939960208774
Model WCE-Sensor   = -8.75638074294346
Model WDE-Sensor   = -9.45395266031472
Model Wheel-Sensor = 13.1031783871962
   wheel delta-rpm = 186.544048758552
Model ESA-Sensor   = 0.47756298633463
   esa omega       = 6.57992094525853e-4

Real EPE-Sensor   = 0.472939960208774
Real WCE-Sensor   = 25
```

```
Real WDE-Sensor    = 11.1190589701237
Real Wheel-Sensor = -15.4110157325915
  wheel delta-rpm = 131.603725387073
Real ESA Sensor    = 0.478652299563926
  esa omega        = 4.64202484566667e-4

DONE
[41] (diagnose)
bad-sensor =M-WCE-SENSOR
suspect list =(M-WCE-A M-EPE-A)
checking suspect: M-WCE-A
checking fault: HIGH
disc sensor from all= M-WCE-SENSOR
checking fault: LOW
disc sensor from all= M-WCE-SENSOR
checking fault: ZERO
disc sensor from all= M-WCE-SENSOR
checking fault: LATCHUP
disc sensor from all= M-WDE-SENSOR
checking fault: LATCHDOWN
disc sensor from all= M-WCE-SENSOR
suspect list =(M-EPE-A)
checking suspect: M-EPE-A
checking fault: HIGH
disc sensor from all= M-WCE-SENSOR
checking fault: LOW
disc sensor from all= M-WCE-SENSOR
checking fault: ZERO
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHUP
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHDOWN
disc sensor from all= M-EPE-SENSOR
suspect list =()
*** Fault detected: Cause Unknown! ***
()
[42] (sp)

Model EPE-Sensor    = 0.467164644374392
Model WCE-Sensor    = 7.82396905603094
Model WDE-Sensor    = 2.84440235755722
Model Wheel-Sensor = -3.94234166757431
   wheel delta-rpm = 223.655320367624
Model ESA-Sensor    = 0.438149559793175
   esa omega        = 7.88893742147841e-4

Real EPE-Sensor    = 0.467164644374392
Real WCE-Sensor    = 25
Real WDE-Sensor    = 14.1397289508637
Real Wheel-Sensor = -19.5976643258971
  wheel delta-rpm = 94.3734536117936
Real ESA Sensor    = 0.474065152516732
  esa omega        = 3 32881090674921e-4

DONE
[43] (fix r-wce-a)
ON
[44] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :
```

```
Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-A
   STATUS : ON
   STATE : 25
   INPUT-LIST : (R-EPE-A)
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0.478681459056942
   YN-1 : 0.564833866564236
   YN-2 : 0.617228472790691
   LIMIT : 25
   TOLERANCE : 1.e-5
()
[45] (sp)

Model EPE-Sensor    = 0.462687588856331
Model WCE-Sensor    = -21.5271636204109
Model WDE-Sensor    = -13.5821135434337
Model Wheel-Sensor = 18.8248093711992
   wheel delta-rpm = 254.995517684589
Model ESA-Sensor    = 0.428539172640725
   esa omega        = 8.99439225708857e-4

Real EPE-Sensor    = 0.462687588856331
Real WCE-Sensor    = -8.16822270319648
Real WDE-Sensor    = -3.7181200050228
Real Wheel-Sensor = 5.1533143269616
  wheel delta-rpm = 104.25696133443
Real ESA Sensor    = 0.470054852747156
  esa omega        = 3.67742936930319e-4

DONE
[46] (sp)

Model EPE-Sensor    = 0.458773536281224
Model WCE-Sensor    = -19.6529266742498
Model WDE-Sensor    = -14.2770982914002
Model Wheel-Sensor = 19.7880582318806
   wheel delta-rpm = 295.052081850433
Model ESA-Sensor    = 0.417417149297213
   esa omega        = 0.00104072972910683

Real EPE-Sensor    = 0.458773536281224
Real WCE-Sensor    = -7.72115899754946
Real WDE-Sensor    = -4.41831933995724
Real Wheel-Sensor = 6.12379060518073
  wheel delta-rpm = 115.960872068542
Real ESA Sensor    = 0.465609461768215
  esa omega        = 4.09025844583135e-4

DONE
[47] (sp)

Model EPE-Sensor    = 0.454434834685778
Model WCE-Sensor    = -14.4467512853061
Model WDE-Sensor    = -12.400604544565
Model Wheel-Sensor = 17.1872378987671
   wheel delta-rpm = 330.728253073301
Model ESA-Sensor    = 0.404756170914093
   esa omega        = 0.00116656938351458

Real EPE-Sensor    = 0.454434834685778
Real WCE-Sensor    = -7.58719730095281
```

```
Real WDE-Sensor    = -5.05429330090393
Real Wheel-Sensor = 7.00525051505285
  wheel delta-rpm = 129.496658676107
Real ESA Sensor    = 0.460653247259313
  esa omega        = 4.56770281568605e-4

DONE
[48] (sp)

Model EPE-Sensor    = 0.44959756932509
Model WCE-Sensor    = -10.8810604113251
Model WDE-Sensor    = -10.7494381488415
Model Wheel-Sensor = 14.8987212742943
   wheel delta-rpm = 361.511212037846
Model ESA-Sensor    = 0.390755163384707
   esa omega        = 0.0012751493343604

Real EPE-Sensor    = 0.44959756932509
Real WCE-Sensor    = -7.48853863349113
Real WDE-Sensor    = -5.54358412877156
Real Wheel-Sensor = 7.6834076024[illegible]38
  wheel delta-rpm = 144.486621047792
Real ESA Sensor    = 0.455119501[illegible]8939
  esa omega        = 5.096439958e5301e-4

DONE
[49] (sp)

Model EPE-Sensor    = 0.444196633746444
Model WCE-Sensor    = -8.92097782532382
Model WDE-Sensor    = -9.6218580131477
Model Wheel-Sensor = 13.3358952062227
   wheel delta-rpm = 388.813092296559
Model ESA-Sensor    = 0.375583778501739
   esa omega        = 0.00137145056452817

Real EPE-Sensor    = 0.444196633746444
Real WCE-Sensor    = -7.36736034165554
Real WDE-Sensor    = -5.89139221558405
Real Wheel-Sensor = 8.16546961079949
  wheel delta-rpm = 160.527908837168
Real ESA Sensor    = 0.448957693691164
  esa omega        = 5.66226023658023e-4

DONE
[50] (sp)

Model EPE-Sensor    = 0.438182709042576
Model WCE-Sensor    = -7.89739897850404
Model WDE-Sensor    = -8.86267312007939
Model Wheel-Sensor = 12.28366494443
   wheel delta-rpm = 413.778937084957
Model ESA-Sensor    = 0.359358336010273
   esa omega        = 0.0014595119559971

Real EPE-Sensor    = 0.438182709042576
Real WCE-Sensor    = -7.21846521267016
Real WDE-Sensor    = -6.12040634085861
Real Wheel-Sensor = 8.48288318843004
  wheel delta-rpm = 177.274270520968
Real ESA Sensor    = 0.442132495217498
  esa omega        = 6.252950407258O1e-4

DONE
[51] (sp)
```

```
Model EPE-Sensor    = 0.431521315332279
Model WCE-Sensor    = -7.33358127181175
Model WDE-Sensor    = -8.31578022137679
Model Wheel-Sensor = 11.5256713868282
   wheel delta-rpm = 437.095101386718
Model ESA-Sensor    = 0.34215840319032
   esa omega        = 0.00154175447130286

Real EPE-Sensor    = 0.431521315332279
Real WCE-Sensor    = -7.04609778629477
Real WDE-Sensor    = -6.25288354895924
Real Wheel-Sensor = 8.66649659885751
  wheel delta-rpm = 194.442896090122
Real ESA Sensor    = 0.434621275128168
  esa omega        = 6.85853498492519e-4

DONE
[52] (fail r-epe-a 'zero)
ZERO
[53] (sp)

Model EPE-Sensor    = 0.424190364525092
Model WCE-Sensor    = -6.97344619317545
Model WDE-Sensor    = -7.8845207423793
Model Wheel-Sensor = 10.9279457489377
   wheel delta-rpm = 459.139346560191
Model ESA-Sensor    = 0.324042451020106
   esa omega        = 0.00161951057850899

Real EPE-Sensor    = 0
Real WCE-Sensor    = 17.4805049324202
Real WDE-Sensor    = 8.24971540243142
Real Wheel-Sensor = -11.4341055477699
  wheel delta-rpm = 176.728785619577
Real ESA Sensor    = 0.427067333453463
  esa omega        = 6.23370965660491e-4

DONE
[54] (diagnose)
bad-sensor =M-EPE-SENSOR
suspect list =(M-EPE-A)
checking suspect: M-EPE-A
checking fault: HIGH
disc sensor from all= M-EPE-SENSOR
checking fault: LOW
disc sensor from all= M-EPE-SENSOR
checking fault: ZERO
disc sensor from all= M-WCE-SENSOR
checking fault: LATCHUP
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHDOWN
disc sensor from all= M-EPE-SENSOR
suspect list =()
*** Fault detected: Cause Unknown! ***
()
[55] (transcript-off)
```

Appendix J: Program Run - Modified Model

This program run tests the model-based reasoner without filters in the WCE and WDE of the real and model-world pitch channel systems. The (reset-system2) invocation loads the MPCHL2.S and RPCHL2.S which instantiate the WCE and WDE as simple amplifiers, rather that as objects that use difference equations to model the compensation filters.

```
[3] (reset-system2)
OK
[4] (set-pitch .1)

Model EPE-Sensor    = 0.0976
Model WCE-Sensor    = -7.808
Model WDE-Sensor    = -7.808
Model Wheel-Sensor = 10.821888
   wheel delta-rpm = 10.821888
Model ESA-Sensor    = 0.0998906378754519
   esa omega        = 3.81717712209661e-5

Real EPE-Sensor    = 0.0976
Real WCE-Sensor    = -7.808
Real WDE-Sensor    = -7.808
Real Wheel-Sensor = 10.821888
  wheel delta-rpm = 10.821888
Real ESA Sensor    = 0.0998906378754519
  esa omega        = 3.81717712209661e-5

DONE
[5] (describe r-epe-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class EPE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-EPE-A
   STATUS : ON
   STATE : 0.0976
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (R-WCE-A R-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 1.e-4
()
[6] (fail r-epe-a)

[VM ERROR encountered!]
Invalid argument count: Function expected  2 argument(s)
but was called with  1 as follows:
(#<PROCEDURE FAIL> #<ENVIRONMENT>)

[Inspect] Quit
[7] (fail r-epe-a 'latchup)
```

```
LATCHUP
[8] (describe r-epe-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class EPE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-EPE-A
   STATUS : LATCHUP
   STATE : 0.0976
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (R-WCE-A R-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 1.e-4
()
[9] (sp)

Model EPE-Sensor    = 0.0974932625664411
Model WCE-Sensor    = -7.808
Model WDE-Sensor    = -7.808
Model Wheel-Sensor = 10.821888
   wheel delta-rpm = 10.821888
Model ESA-Sensor    = 0.0998906378754519
   esa omega        = 3.81717712209661e-5

Real EPE-Sensor    = 5
Real WCE-Sensor    = -25
Real WDE-Sensor    = -25.
Real Wheel-Sensor = 34.65
  wheel delta-rpm = 45.471888
Real ESA Sensor    = 0.0993217531353592
  esa omega        = 1.60391837886457e-4

DONE
[10] (diagnose)
The culprit is: M-EPE-A
()
[11] (epe-b-on)
()
[12] (sp)

Model EPE-Sensor    = 0.0969380310601105
Model WCE-Sensor    = -7.75504248480884
Model WDE-Sensor    = -7.75504248480884
Model Wheel-Sensor = 10.7484888839451
   wheel delta-rpm = 56.2203768839451
Model ESA-Sensor    = 0.0982940875249025
   esa omega        = 1.98304710265939e-4

Real EPE-Sensor    = 0.0969380310601105
Real WCE-Sensor    = -7.75504248480884
Real WDE-Sensor    = -7.75504248480884
Real Wheel-Sensor = 10.7484888839451
  wheel delta-rpm = 56.2203768839451
Real ESA Sensor    = 0.0982940875249025
  esa omega        = 1.98304710265939e-4

DONE
[13] (diagnose)
```

```
No fault found!
()
[14] (fail r-wce-a 'zero)
ZERO
[15] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class AMPLIFIER

 Class Variables :

Instance Variables :
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   INPUT-LIST : (R-EPE-B)
   STATE : -7.75504248480884
   STATUS : ZERO
   NAME : R-WCE-A
   GAIN : -80
   LIMIT : 25
   TOLERANCE : 0.01
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
()
[16] (sp)

Model EPE-Sensor    = 0.0959350294243049
Model WCE-Sensor    = -7.67480235394439
Model WDE-Sensor    = -7.67480235394439
Model Wheel-Sensor = 10.6372760625669
   wheel delta-rpm = 66.857652946512
Model ESA-Sensor    - 0.0970503050326564
   esa omega        = 2.35825304479681e-4

Real EPE-Sensor    = 0.0959350294243049
Real WCE-Sensor    = 0
Real WDE-Sensor    = 0.
Real Wheel-Sensor = 0.
  wheel delta-rpm = 56.2203768839451
Real ESA Sensor    = 0.0971578015350787
  esa omega        = 1.98304710265939e-4

DONE
[17] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class AMPLIFIER

 Class Variables :

Instance Variables :
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   INPUT-LIST : (R-EPE-B)
   STATE : 0
   STATUS : ZERO
   NAME : R-WCE-A
   GAIN : -80
   LIMIT : 25
   TOLERANCE : 0.01
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
()
[18] (diagnose)
The culprit is: M-WCE-A
```

```
()
[19] (wce-b-on)
()
[20] (sp)

Model EPE-Sensor    = 0.0948260142982368
Model WCE-Sensor    = -7.58608114385895
Model WDE-Sensor    = -7.58608114385895
Model Wheel-Sensor = 10.5143084653885
   wheel delta-rpm = 98.6465135370343
Model ESA-Sensor    = 0.0800820743996125
   esa omega        = 3.47953346632456e-4

Real EPE-Sensor    = 0.0948260142982368
Real WCE-Sensor    = -7.58608114385895
Real WDE-Sensor    = -7.58608114385895
Real Wheel-Sensor = 10.5143084653885
  wheel delta-rpm = 77.3719614119005
Real ESA Sensor    = 0.094456486212241
  esa omeqa        = 2.72912158204972e-4

DONE
[21] (fail r-wde-a 'low)
LOW
[22] (sp)

Model EPE-Sensor    = 0.0921895305431472
Model WCE-Sensor    = -7.37516244345178
Model WDE-Sensor    = -7.37516244345178
Model Wheel-Sensor = 10.2219751466242
   wheel delta-rpm = 108.868488683659
Model ESA-Sensor    = 0.0779850021033133
   esa omega        = 3.84009060452756e-4

Real EPE-Sensor    = 0.0921895305431472
Real WCE-Sensor    = -7.37516244345178
Real WDE-Sensor    = -12.3751624434518
Real Wheel-Sensor = 17.1519751466242
  wheel delta-rpm = 94.5239365585246
Real ESA Sensor    = 0.092719367827432
  esa omega        = 3.3341188535837e-4

DONE
[23] (diagnose)
The culprit is: M-WDE-A
()
[24] (sp)

Model EPE-Sensor    = 0.0904941029995736
Model WCE-Sensor    = -7.23952823996589
Model WDE-Sensor    = -7.23952823996589
Model Wheel-Sensor = 10.0339861405927
   wheel delta-rpm = 125.140375410748
Model ESA-Sensor    = 0.0466467424402125
   esa omega        = 4.41404474033079e-4

Real EPE-Sensor    = 0.0904941029995736
Real WCE-Sensor    = -7.23952823996589
Real WDE-Sensor    = -12.2395282399659
Real Wheel-Sensor = 16.9639861405927
  wheel delta-rpm = 111.487922699117
Real ESA Sensor    = 0.0906374857556056
  esa omega        = 3.93248523656081e-4

DONE
```

```
[25] (fix-r-wde-a)

[VM ERROR encountered!] Variable not defined in current environment
FIX-R-WDE-A

[Inspect] Quit
[26] (fix r-wde-a)
ON
[27] (sp)

Model EPE-Sensor    = 0.0884621860974711
Model WCE-Sensor    = -7.07697488779769
Model WDE-Sensor    = -7.07697488779769
Model Wheel-Sensor = 9.80868719448759
   wheel delta-rpm = 134.949062605236
Model ESA-Sensor    = 0.0440183717239436
   esa omega        = 4.76002407736144e-4

Real EPE-Sensor    = 0.0884621860974711
Real WCE-Sensor    = -7.07697488779769
Real WDE-Sensor    = -7.07697488779769
Real Wheel-Sensor = 9.80868719448759
  wheel delta-rpm = 121.296609893605
Real ESA Sensor    = 0.088285048634997
  esa omega        = 4.27846457359147e-4

DONE
[28] (fail r-wce-b 'zero)
ZERO
[29] (sp)

Model EPE-Sensor    = 0.0861662074677571
Model WCE-Sensor    = -6.89329659742057
Model WDE-Sensor    = -6.89329659742057
Model Wheel-Sensor = 9.5541090840249
   wheel delta-rpm = 144.50317168926
Model ESA-Sensor    = 0.04119432752277
   esa omega        = 5.097023745234l4e-4

Real EPE-Sensor    = 0.0861662074677571
Real WCE-Sensor    = 0
Real WDE-Sensor    = 0.
Real Wheel-Sensor = 0.
  wheel delta-rpm = 121.296609893605
Real ESA Sensor    = 0.0858334884343291
  esa omega        = 4.27846457359147e-4

DONE
[30] (diagnose)
The culprit is: M-WCE-B
()
[31] (fail r-wde-a 'latchdown)
LATCHDOWN
[32] (sp)

Model EPE-Sensor    = 0.0837734847119052
Model WCE-Sensor    = -6.70187877695241
Model WDE-Sensor    = -6.70187877695241
Model Wheel-Sensor = 9.28880398485605
   wheel delta-rpm = 172.900193842166
Model ESA-Sensor    = 0.0104649160194273
   esa omega        = 6.09866470934079e-4

Real EPE-Sensor    = 0.0837734847119052
Real WCE-Sensor    = 0
```

```
Real WDE-Sensor    = -25
Real Wheel-Sensor = 34.65
  wheel delta-rpm = 155.946609893605
Real ESA Sensor    = 0.0830317677426645
  esa omega        = 5.50066524024638e-4

DONE
[33] (diagnose)
*** Fault detected: Cause Unknown! ***
()
[34] (wce-b-on)
()
[35] (wde-a-on)
()
[36] (sp)

Model EPE-Sensor    = 0.0810390053168406
Model WCE-Sensor    = -6.48312042534725
Model WDE-Sensor    = -6.48312042534725
Model Wheel-Sensor = 8.98560490953128
   wheel delta-rpm = 165.81340672141
Model ESA-Sensor    = -0.0246590252098984
   esa omega        = 5.84869426364295e-4

Real EPE-Sensor    = 0.0810390053168406
Real WCE-Sensor    = -6.48312042534725
Real WDE-Sensor    = -6.48312042534725
Real Wheel-Sensor = 8.98560490953128
  wheel delta-rpm = 174.221018787992
Real ESA Sensor    = 0.0763555920925125
  esa omega        = 6.14525383284218e-4

DONE
[37] (diagnose)
No fault found!
()
[38] (fail r-wheel-a 'zero)
ZERO
[39] (sp)

Model EPE-Sensor    = 0.0745230578822922
Model WCE-Sensor    = -5.96184463058337
Model WDE-Sensor    = -5.96184463058337
Model Wheel-Sensor = 8.26311665798856
   wheel delta-rpm = 174.076523379398
Model ESA-Sensor    = -0.0280938311206894
   esa omega        = 6.14015703292263e-4

Real EPE-Sensor    = 0.0745230578822922
Real WCE-Sensor    = -5.96184463058337
Real WDE-Sensor    = -5.96184463058337
Real Wheel-Sensor = 0
  wheel delta-rpm = 174.221018787992
Real ESA Sensor    = 0.0728343616462939
  esa omega        = 6.14525383284218e-4

DONE
[40] (diagnose)
The suspects are: (M-WHEEL-SENSOR M-WHEEL-A)
()
[41] (fix r-wheel-a)
ON
[42] (sp)

Model EPE-Sensor    = 0.0710863369667828
```

```
Model WCE-Sensor    = -5.68690695734263
Model WDE-Sensor    = -5.68690695734263
Model Wheel-Sensor = 7.88205304287688
   wheel delta-rpm = 188.624159712218
Model ESA-Sensor    = -0.100832882740029
   esa omega        = 6.94475499092455e-4

Real EPE-Sensor     = 0.0710863369667828
Real WCE-Sensor     = -5.68690695734263
Real WDE-Sensor     = -5.68690695734263
Real Wheel-Sensor = 7.88205304287688
  wheel delta-rpm = 182.103071830869
Real ESA Sensor     = 0.0692334779947779
  esa omega         = 6.42327549182093e-4

DONE
[43] (diagnose)
No fault found!
()
[44] (fail r-wce-sensor 'zero)
ZERO
[45] (sp)

Model EPE-Sensor    = 0.0675718745229032
Model WCE-Sensor    = -5.40574996183226
Model WDE-Sensor    = -5.40574996183226
Model Wheel-Sensor = 7.49236944709951
   wheel delta-rpm = 196.116529159317
Model ESA-Sensor    = -0.104887942552263
   esa omega        = 7.20903143920939e-4

Real EPE-Sensor     = 0.0675718745229032
Real WCE-Sensor     = 0
Real WDE-Sensor     = -5.40574996183226
Real Wheel-Sensor = 7.49236944709951
  wheel delta-rpm = 189.595441277969
Real ESA Sensor     = 0.0654772259355309
  esa omega         = 6.68755194010577e-4

DONE
[46] (diagnose)
The culprit is: M-WCE-SENSOR
()
[47] (transcript-off)
```

Appendix K: Program Run - Modified Model -Verbose

This listing is a diagnostic run, similar to Appendix C, except that the debugging *writelns* have been left in the code. Although lengthy, this listing has been included here to enable the ready to follow through the operation of the propogation mechanism, and the diagnostic reasoning algorithm.

```
[3] (reset-system2)
OK
[4] (set-pitch .1)
Change COMMAND-1 to 0.1.
Change M-EPE-A to 0.0976.
Change M-WCE-A to -7.808.
Change M-WDE-A to -7.808.
Change M-WHEEL-A to 10.821888.
Change M-ESA-A to 0.0998906378754519.
Change M-ESA-SENSOR to 0.0998906378754519.
Change M-WHEEL-SENSOR to 10.821888.
Change M-WDE-SENSOR to -7.808.
Change M-WCE-SENSOR to -7.808.
Change M-EPE-SENSOR to 0.0976.
Change M-EPE-A to ().
Change R-EPE-A to 0.0976.
Change R-WCE-A to -7.808.
Change R-WDE-A to -7.808.
Change R-WHEEL-A to 10.821888.
Change R-ESA-A to 0.0998906378754519.
Change R-ESA-SENSOR to 0.0998906378754519.
Change R-WHEEL-SENSOR to 10.821888.
Change R-WDE-SENSOR to -7.808.
Change R-WCE-SENSOR to -7.808.
Change R-EPE-SENSOR to 0.0976.
Change R-EPE-A to ().

Model EPE-Sensor    = 0.0976
Model WCE-Sensor    = -7.808
Model WDE-Sensor    = -7.808
Model Wheel-Sensor = 10.821888
   wheel delta-rpm = 10.821888
Model ESA-Sensor    = 0.0998906378754519
   esa omega        = 3.81717712209661e-5

Real EPE-Sensor    = 0.0976
Real WCE-Sensor    = -7.808
Real WDE-Sensor    = -7.808
Real Wheel-Sensor = 10.821888
  wheel delta-rpm = 10.821888
Real ESA Sensor    = 0.0998906378754519
  esa omega        = 3.81717712209661e-5

DONE
[5] (describe r-epe-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class EPE
```

```
 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-EPE-A
   STATUS : ON
   STATE : 0.0976
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (R-WCE-A R-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 1.e-4
()
[6] (fail r-epe-a)

[VM ERROR encountered!]
Invalid argument count: Function expected  2 argument(s)
but was called with  1 as follows:
(#<PROCEDURE FAIL> #<ENVIRONMENT>)

[Inspect] Quit
[7] (fail r-epe-a 'latchup)
LATCHUP
[8] (describe r-epe-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class EPE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-EPE-A
   STATUS : LATCHUP
   STATE : 0.0976
   INPUT-LIST : (COMMAND-1)
   OUTPUT-LIST : (R-WCE-A R-EPE-SENSOR)
   GAIN : 0.976
   LIMIT : 5
   TOLERANCE : 1.e-4
()
[9] (sp)
Change COMMAND-1 to 0.0998906378754519.
Change M-EPE-A to 0.0974932625664411.
Change M-EPE-SENSOR to 0.0974932625664411.
Change M-EPE-A to ().
Change R-EPE-A to 5.
Change R-WCE-A to -25.
Change R-WDE-A to -25..
Change R-WHEEL-A to 34.65.
Change R-ESA-A to 0.0993217531353592.
Change R-ESA-SENSOR to 0.0993217531353592.
Change R-WHEEL-SENSOR to 34.65.
Change R-WDE-SENSOR to -25..
Change R-WCE-SENSOR to -25.
Change R-EPE-SENSOR to 5.
Change R-EPE-A to ().

Model EPE-Sensor   = 0.0974932625664411
Model WCE-Sensor   = -7.808
Model WDE-Sensor   = -7.808
Model Wheel-Sensor = 10.821888
```

```
   wheel delta-rpm = 10.821888
Model ESA-Sensor   = 0.0998906378754519
   esa omega       = 3.81717712209661e-5

Real EPE-Sensor    = 5
Real WCE-Sensor    = -25
Real WDE-Sensor    = -25.
Real Wheel-Sensor = 34.65
  wheel delta-rpm = 45.471888
Real ESA Sensor    = 0.0993217531353592
  esa omega        = 1.60391837886457e-4

DONE
[10] (diagnose)
bad-sensor =M-EPE-SENSOR
suspect list =(M-EPE-A)
checking suspect: M-EPE-A
checking fault: HIGH
Change M-EPE-A to 5.
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to 0.0993217531353592.
Change M-ESA-SENSOR to 0.0993217531353592.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= ()
suspect list =()
The culprit is: M-EPE-A
()
[11] (epe-b-on)
()
[12] (sp)
Change COMMAND-1 to 0.0993217531353592.
Change M-EPE-A to 0.0969380310601105.
Change M-WCE-A to -7.75504248480884.
Change M-WDE-A to -7.75504248480884.
Change M-WHEEL-A to 10.7484888839451.
Change M-ESA-A to 0.0982940875249025.
Change M-ESA-SENSOR to 0.0982940875249025.
Change M-WHEEL-SENSOR to 10.7484888839451.
Change M-WDE-SENSOR to -7.75504248480884.
Change M-WCE-SENSOR to -7.75504248480884.
Change M-EPE-SENSOR to 0.0969380310601105.
Change R-EPE-A to ().
Change R-EPE-A to 0.0969380310601105.
Change R-WCE-A to -7.75504248480884.
Change R-WDE-A to -7.75504248480884.
Change R-WHEEL-A to 10.7484888839451.
Change R-ESA-A to 0.0982940875249025.
Change R-ESA-SENSOR to 0.0982940875249025.
Change R-WHEEL-SENSOR to 10.7484888839451.
Change R-WDE-SENSOR to -7.75504248480884.
Change R-WCE-SENSOR to -7.75504248480884.
Change R-EPE-SENSOR to 0.0969380310601105.

Model EPE-Sensor   = 0.0969380310601105
Model WCE-Sensor   = -7.75504248480884
Model WDE-Sensor   = -7.75504248480884
Model Wheel-Sensor = 10.7484888839451
   wheel delta-rpm = 56.2203768839451
Model ESA-Sensor   = 0.0982940875249025
   esa omega       = 1.98304710265939e-4
```

```
Real EPE-Sensor    = 0.0969380310601105
Real WCE-Sensor    = -7.75504248480884
Real WDE-Sensor    = -7.75504248480884
Real Wheel-Sensor = 10.7484888839451
  wheel delta-rpm = 56.2203768839451
Real ESA Sensor    = 0.0982940875249025
  esa omega        = 1.98304710265939e-4

DONE
[13] (diagnose)
bad-sensor =()
No fault found!
()
[14] (fail r-wce-a 'zero)
ZERO
[15] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class AMPLIFIER

 Class Variables :

Instance Variables :
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   INPUT-LIST : (R-EPE-B)
   STATE : -7.75504248480884
   STATUS : ZERO
   NAME : R-WCE-A
   GAIN : -80
   LIMIT : 25
   TOLERANCE : 0.01
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
()
[16] (sp)
Change COMMAND-1 to 0.0982940875249025.
Change M-EPE-A to 0.0959350294243049.
Change M-WCE-A to -7.67480235394439.
Change M-WDE-A to -7.67480235394439.
Change M-WHEEL-A to 10.6372760625669.
Change M-ESA-A to 0.0970503050326564.
Change M-ESA-SENSOR to 0.0970503050326564.
Change M-WHEEL-SENSOR to 10.6372760625669.
Change M-WDE-SENSOR to -7.67480235394439.
Change M-WCE-SENSOR to -7.67480235394439.
Change M-EPE-SENSOR to 0.0959350294243049.
Change R-EPE-A to ().
Change R-WCE-A to 0.
Change R-WDE-A to 0..
Change R-WHEEL-A to 0..
Change R-ESA-A to 0.0971578015350787.
Change R-ESA-SENSOR to 0.0971578015350787.
Change R-WHEEL-SENSOR to 0..
Change R-WDE-SENSOR to 0..
Change R-WCE-SENSOR to 0.
Change R-EPE-A to 0.0959350294243049.
Change R-WCE-A to 0.
Change R-EPE-SENSOR to 0.0959350294243049.

Model EPE-Sensor    = 0.0959350294243049
Model WCE-Sensor    = -7.67480235394439
Model WDE-Sensor    = -7.67480235394439
Model Wheel-Sensor = 10.6372760625669
   wheel delta-rpm = 66.857652946512
```

```
Model ESA-Sensor    = 0.0970503050326564
   esa omega        = 2.35825304479681e-4

Real EPE-Sensor    = 0.0959350294243049
Real WCE-Sensor    = 0
Real WDE-Sensor    = 0.
Real Wheel-Sensor  = 0.
  wheel delta-rpm  = 56.2203768839451
Real ESA Sensor    = 0.0971578015350787
  esa omega        = 1.98304710265939e-4

DONE
[17] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class AMPLIFIER

 Class Variables :

Instance Variables :
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   INPUT-LIST : (R-EPE-B)
   STATE : 0
   STATUS : ZERO
   NAME : R-WCE-A
   GAIN : -80
   LIMIT : 25
   TOLERANCE : 0.01
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
()
[18] (diagnose)
bad-sensor =M-WCE-SENSOR
suspect list =(M-WCE-A M-EPE-B)
checking suspect: M-WCE-A
checking fault: HIGH
Change M-WCE-A to -2.67480235394439.
Change M-WDE-A to -2.67480235394439.
Change M-WHEEL-A to 3.70727606256693.
Change M-ESA-A to 0.0956615616337647.
Change M-ESA-SENSOR to 0.0956615616337647.
Change M-WHEEL-SENSOR to 3.70727606256693.
Change M-WDE-SENSOR to -2.67480235394439.
Change M-WCE-SENSOR to -2.67480235394439.
disc sensor from all= M-WCE-SENSOR
checking fault: LOW
Change M-WCE-A to -12.6748023539444.
Change M-WDE-A to -12.6748023539444.
Change M-WHEEL-A to 17.5672760625669.
Change M-ESA-A to 0.0940578252300284.
Change M-ESA-SENSOR to 0.0940578252300284.
Change M-WHEEL-SENSOR to 17.5672760625669.
Change M-WDE-SENSOR to -12.6748023539444.
Change M-WCE-SENSOR to -12.6748023539444.
disc sensor from all= M-WCE-SENSOR
checking fault: ZERO
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to 0.0922765602256703.
Change M-ESA-SENSOR to 0.0922765602256703.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
```

```
disc sensor from all= ()
suspect list =(M-EPE-B)
checking suspect: M-EPE-B
checking fault: HIGH
Change M-EPE-A to 5.
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to 0.0901451347303156.
Change M-ESA-SENSOR to 0.0901451347303156.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LOW
Change M-EPE-A to -4.9040649705757.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 0.0880137092349609.
Change M-ESA-SENSOR to 0.0880137092349609.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -4.9040649705757.
disc sensor from all= M-EPE-SENSOR
checking fault: ZERO
Change M-EPE-A to 0.
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to 0.0862324442306029.
Change M-ESA-SENSOR to 0.0862324442306029.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
Change M-EPE-SENSOR to 0.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHUP
Change M-EPE-A to 5.
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to 0.0841010187352482.
Change M-ESA-SENSOR to 0.0841010187352482.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHDOWN
Change M-EPE-A to -5.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 0.0819695932398935.
Change M-ESA-SENSOR to 0.0819695932398935.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -5.
disc sensor from all= M-EPE-SENSOR
suspect list =()
The culprit is: M-WCE-A
```

```
()
[19] (wce-b-on)
()
[20] (sp)
Change COMMAND-1 to 0.0971578015350787.
Change M-EPE-A to 0.0948260142982368.
Change M-WCE-A to -7.58608114385895.
Change M-WDE-A to -7.58608114385895.
Change M-WHEEL-A to 10.5143084653885.
Change M-ESA-A to 0.0800820743996125.
Change M-ESA-SENSOR to 0.0800820743996125.
Change M-WHEEL-SENSOR to 10.5143084653885.
Change M-WDE-SENSOR to -7.58608114385895.
Change M-WCE-SENSOR to -7.58608114385895.
Change M-EPE-SENSOR to 0.0948260142982368.
Change R-EPE-A to ().
Change R-WCE-A to -7.67480235394439.
Change R-WDE-A to -7.67480235394439.
Change R-WHEEL-A to 10.6372760625669.
Change R-ESA-A to 0.0959140190428325.
Change R-ESA-SENSOR to 0.0959140190428325.
Change R-WHEEL-SENSOR to 10.6372760625669.
Change R-WDE-SENSOR to -7.67480235394439.
Change R-WCE-SENSOR to -7.67480235394439.
Change R-EPE-A to 0.0948260142982368.
Change R-WCE-A to -7.58608114385895.
Change R-WDE-A to -7.58608114385895.
Change R-WHEEL-A to 10.5143084653885.
Change R-ESA-A to 0.094456486212241.
Change R-ESA-SENSOR to 0.094456486212241.
Change R-WHEEL-SENSOR to 10.5143084653885.
Change R-WDE-SENSOR to -7.58608114385895.
Change R-WCE-SENSOR to -7.58608114385895.
Change R-EPE-SENSOR to 0.0948260142982368.

Model EPE-Sensor    = 0.0948260142982368
Model WCE-Sensor    = -7.58608114385895
Model WDE-Sensor    = -7.58608114385895
Model Wheel-Sensor = 10.5143084653885
   wheel delta-rpm = 98.6465135370343
Model ESA-Sensor    = 0.0800820743996125
   esa omega        = 3.47953346632456e-4

Real EPE-Sensor    = 0.0948260142982368
Real WCE-Sensor    = -7.58608114385895
Real WDE-Sensor    = -7.58608114385895
Real Wheel-Sensor = 10.5143084653885
  wheel delta-rpm = 77.3719614119005
Real ESA Sensor    = 0.094456486212241
  esa omega        = 2.72912158204972e-4

DONE
[21] (fail r-wde-a 'low)
LOW
[22] (sp)
Change COMMAND-1 to 0.094456486212241.
Change M-EPE-A to 0.0921895305431472.
Change M-WCE-A to -7.37516244345178.
Change M-WDE-A to -7.37516244345178.
Change M-WHEEL-A to 10.2219751466242.
Change M-ESA-A to 0.0779850021033133.
Change M-ESA-SENSOR to 0.0779850021033133.
Change M-WHEEL-SENSOR to 10.2219751466242.
Change M-WDE-SENSOR to -7.37516244345178.
Change M-WCE-SENSOR to -7.37516244345178.
```

```
Change M-EPE-SENSOR to 0.0921895305431472.
Change R-EPE-A to ().
Change R-EPE-A to 0.0921895305431472.
Change R-WCE-A to -7.37516244345178.
Change R-WDE-A to -12.3751624434518.
Change R-WHEEL-A to 17.1519751466242.
Change R-ESA-A to 0.092719367827432.
Change R-ESA-SENSOR to 0.092719367827432.
Change R-WHEEL-SENSOR to 17.1519751466242.
Change R-WDE-SENSOP to -12.3751624434518.
Change R-WCE-SENSOR to -7.37516244345178.
Change R-EPE-SENSOR to 0.0921895305431472.

Model EPE-Sensor    = 0.0921895305431472
Model WCE-Sensor    = -7.37516244345178
Model WDE-Sensor    = -7.37516244345178
Model Wheel-Sensor = 10.2219751466242
   wheel delta-rpm = 108.868488683659
Model ESA-Sensor    = 0.0779850021033133
   esa omega        = 3.84009060452756e-4

Real EPE-Sensor    = 0.0921895305431472
Real WCE-Sensor    = -7.37516244345178
Real WDE-Sensor    = -12.3751624434518
Real Wheel-Sensor = 17.1519751466242
  wheel delta-rpm = 94.5239365585246
Real ESA Sensor    = 0.092719367827432
  esa omega        = 3.3341188535837e-4

DONE
[23] (diagnose)
bad-sensor =M-WDE-SENSOR
suspect list =(M-WDE-A M-WCE-B M-EPE-B)
checking suspect: M-WDE-A
checking fault: HIGH
Change M-WDE-A to -2.37516244345178.
Change M-WHEEL-A to 3.29197514662416.
Change M-ESA-A to 0.0757513626650232.
Change M-ESA-SENSOR to 0.0757513626650232.
Change M-WHEEL-SENSOR to 3.29197514662416.
Change M-WDE-SENSOR to -2.37516244345178.
disc sensor from all= M-WDE-SENSOR
checking fault: LOW
Change M-WDE-A to -12.3751624434518.
Change M-WHEEL-A to 17.1519751466242.
Change M-ESA-A to 0.0733111239865428.
Change M-ESA-SENSOR to 0.0733111239865428.
Change M-WHEEL-SENSOR to 17.1519751466242.
Change M-WDE-SENSOR to -12.3751624434518.
disc sensor from all= ()
suspect list =(M-WCE-B M-EPE-B)
checking suspect: M-WCE-B
checking fault: HIGH
Change M-WCE-A to -2.37516244345178.
Change M-WDE-A to -2.37516244345178.
Change M-WHEEL-A to 3.29197514662416.
Change M-ESA-A to 0.070664286067872.
Change M-ESA-SENSOR to 0.070664286067872.
Change M-WHEEL-SENSOR to 3.29197514662416.
Change M-WDE-SENSOR to -2.37516244345178.
Change M-WCE-SENSOR to -2.37516244345178.
disc sensor from all= M-WCE-SENSOR
checking fault: LOW
Change M-WCE-A to -12.3751624434518.
Change M-WDE-A to -12.3751624434518.
```

```
Change M-WHEEL-A to 17.1519751466242.
Change M-ESA-A to 0.0678108489090109.
Change M-ESA-SENSOR to 0.0678108489090109.
Change M-WHEEL-SENSOR to 17.1519751466242.
Change M-WDE-SENSOR to -12.3751624434518.
Change M-WCE-SENSOR to -12.3751624434518.
disc sensor from all= M-WCE-SENSOR
checking fault: ZERO
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to 0.0647840800318554.
Change M-ESA-SENSOR to 0.0647840800318554.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
disc sensor from all= M-WCE-SENSOR
checking fault: LATCHUP
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 0.0621074716456964.
Change M-ESA-SENSOR to 0.0621074716456964.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
disc sensor from all= M-WCE-SENSOR
checking fault: LATCHDOWN
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to 0.0594308632595375.
Change M-ESA-SENSOR to 0.0594308632595375.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
disc sensor from all= M-WCE-SENSOR
suspect list =(M-EPE-B)
checking suspect: M-  E-B
checking fault: HIGH
Change M-EPE-A to 5.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LOW
Change M-EPE-A to -4.90781046945685.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 0.0567542548733786.
Change M-ESA-SENSOR to 0.0567542548733786.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -4.90781046945685.
disc sensor from all= M-EPE-SENSOR
checking fault: ZERO
Change M-EPE-A to 0.
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to 0.0544278069782162.
Change M-ESA-SENSOR to 0.0544278069782162.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
```

```
Change M-EPE-SENSOR to 0.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHUP
Change M-EPE-A to 5.
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to 0.0517511985920573.
Change M-ESA-SENSOR to 0.0517511985920573.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHDOWN
Change M-EPE-A to -5.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 0.0490745902058984.
Change M-ESA-SENSOR to 0.0490745902058984.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -5.
disc sensor from all= M-EPE-SENSOR
suspect list =()
The culprit is: M-WDE-A
()
[24] (fix r-wde-a)
ON
[25] (sp)
Change COMMAND-1 to 0.092719367827432.
Change M-EPE-A to 0.0904941029995736.
Change M-WCE-A to -7.23952823996589.
Change M-WDE-A to -7.23952823996589.
Change M-WHEEL-A to 10.0339861405927.
Change M-ESA-A to 0.0466467424402125.
Change M-ESA-SENSOR to 0.0466467424402125.
Change M-WHEEL-SENSOR to 10.0339861405927.
Change M-WDE-SENSOR to -7.23952823996589.
Change M-WCE-SENSOR to -7.23952823996589.
Change M-EPE-SENSOR to 0.0904941029995736.
Change R-EPE-A to ().
Change R-EPE-A to 0.0904941029995736.
Change R-WCE-A to -7.23952823996589.
Change R-WDE-A to -7.23952823996589.
Change R-WHEEL-A to 10.0339861405927.
Change R-ESA-A to 0.0907075178538049.
Change R-ESA-SENSOR to 0.0907075178538049.
Change R-WHEEL-SENSOR to 10.0339861405927.
Change R-WDE-SENSOR to -7.23952823996589.
Change R-WCE-SENSOR to -7.23952823996589.
Change R-EPE-SENSOR to 0.0904941029995736.

Model EPE-Sensor    = 0.0904941029995736
Model WCE-Sensor    = -7.23952823996589
Model WDE-Sensor    = -7.23952823996589
Model Wheel-Sensor = 10.0339861405927
   wheel delta-rpm = 125.140375410748
Model ESA-Sensor    = 0.0466467424402125
   esa omega        = 4.41404474033079e-4

Real EPE-Sensor    = 0.0904941029995736
Real WCE-Sensor    = -7.23952823996589
```

```
Real WDE-Sensor    = -7.23952823996589
Real Wheel-Sensor = 10.0339861405927
  wheel delta-rpm = 104.557922699117
Real ESA Sensor    = 0.0907075178538049
  esa omega        = 3.68804510322983e-4

DONE
[26] (diagnose)
bad-sensor =()
No fault found!
()
[27] (fail r-wce-b 'zero)
ZERO
[28] (sp)
Change COMMAND-1 to 0.0907075178538049.
Change M-EPE-A to 0.0885305374253136.
Change M-WCE-A to -7.08244299402509.
Change M-WDE-A to -7.08244299402509.
Change M-WHEEL-A to 9.81626598971878.
Change M-ESA-A to 0.0440182951353532.
Change M-ESA-SENSOR to 0.0440182951353532.
Change M-WHEEL-SENSOR to 9.81626598971878.
Change M-WDE-SENSOR to -7.08244299402509.
Change M-WCE-SENSOR to -7.08244299402509.
Change M-EPE-SENSOR to 0.0885305374253136.
Change R-EPE-A to ().
Change R-WCE-A to 0.
Change R-WDE-A to 0..
Change R-WHEEL-A to 0..
Change R-ESA-A to 0.0885942680096543.
Change R-ESA-SENSOR to 0.0885942680096543.
Change R-WHEEL-SENSOR to 0..
Change R-WDE-SENSOR to 0..
Change R-WCE-SENSOR to 0.
Change R-EPE-A to 0.0885305374253136.
Change R-WCE-A to 0.
Change R-EPE-SENSOR to 0.0885305374253136.

Model EPE-Sensor    = 0.0885305374253136
Model WCE-Sensor    = -7.08244299402509
Model WDE-Sensor    = -7.08244299402509
Model Wheel-Sensor = 9.81626598971878
   wheel delta-rpm = 134.956641400467
Model ESA-Sensor    = 0.0440182951353532
   esa omega        = 4.7602914022845e-4

Real EPE-Sensor    = 0.0885305374253136
Real WCE-Sensor    = 0
Real WDE-Sensor    = 0.
Real Wheel-Sensor = 0.
  wheel delta-rpm = 104.557922699117
Real ESA Sensor    = 0.0885942680096543
  esa omega        = 3.68804510322983e-4

DONE
[29] (diagnose)
bad-sensor =M-WCE-SENSOR
suspect list =(M-WCE-B M-EPE-B)
checking suspect: M-WCE-B
checking fault: HIGH
Change M-WCE-A to -2.08244299402509.
Change M-WDE-A to -2.08244299402509.
Change M-WHEEL-A to 2.88626598971878.
Change M-ESA-A to 0.0412614805913938.
Change M-ESA-SENSOR to 0.0412614805913938.
```

```
Change M-WHEEL-SENSOR to 2.88626598971878.
Change M-WDE-SENSOR to -2.08244299402509.
Change M-WCE-SENSOR to -2.08244299402509.
disc sensor from all= M-WCE-SENSOR
checking fault: LOW
Change M-WCE-A to -12.0824429940251.
Change M-WDE-A to -12.0824429940251.
Change M-WHEEL-A to 16.7462659897188.
Change M-ESA-A to 0.0383062667101348.
Change M-ESA-SENSOR to 0.0383062667101348.
Change M-WHEEL-SENSOR to 16.7462659897188.
Change M-WDE-SENSOR to -12.0824429940251.
Change M-WCE-SENSOR to -12.0824429940251.
disc sensor from all= M-WCE-SENSOR
checking fault: ZERO
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to 0.0351818210620269.
Change M-ESA-SENSOR to 0.0351818210620269.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
disc sensor from all= ()
suspect list =(M-EPE-B)
checking suspect: M-EPE-B
checking fault: HIGH
Change M-EPE-A to 5.
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to 0.0317072149229223.
Change M-ESA-SENSOR to 0.0317072149229223.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LOW
Change M-EPE-A to -4.91146946257469.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 0.0282326087838177.
Change M-ESA-SENSOR to 0.0282326087838177.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -4.91146946257469.
disc sensor from all= M-EPE-SENSOR
checking fault: ZERO
Change M-EPE-A to 0.
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to 0.0251081631357097.
Change M-ESA-SENSOR to 0.0251081631357097.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
Change M-EPE-SENSOR to 0.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHUP
Change M-EPE-A to 5.
Change M-WCE-A to -25.
```

```
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to 0.0216335569966051.
Change M-ESA-SENSOR to 0.0216335569966051.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHDOWN
Change M-EPE-A to -5.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 0.0181589508575005.
Change M-ESA-SENSOR to 0.0181589508575005.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -5.
disc sensor from all= M-EPE-SENSOR
suspect list =()
The culprit is: M-WCE-B
()
[30] (fail r-wde-a 'latchdown)
LATCHDOWN
[31] (sp)
Change COMMAND-1 to 0.0885942680096543.
Change M-EPE-A to 0.0864680055774226.
Change M-WCE-A to -6.91744044619381.
Change M-WDE-A to -6.91744044619381.
Change M-WHEEL-A to 9.58757245842461.
Change M-ESA-A to 0.0149376166356693.
Change M-ESA-SENSOR to 0.0149376166356693.
Change M-WHEEL-SENSOR to 9.58757245842461.
Change M-WDE-SENSOR to -6.91744044619381.
Change M-WCE-SENSOR to -6.91744044619381.
Change M-EPE-SENSOR to 0.0864680055774226.
Change R-EPE-A to ().
Change R-WCE-A to 0.
Change R-WDE-A to -25.
Change R-WHEEL-A to 34.65.
Change R-ESA-A to 0.0861308576745069.
Change R-ESA-SENSOR to 0.0861308576745069.
Change R-WHEEL-SENSOR to 34.65.
Change R-WDE-SENSOR to -25.
Change R-EPE-A to 0.0864680055774226.
Change R-WCE-A to 0.
Change R-WDE-A to -25.
Change R-EPE-SENSOR to 0.0864680055774226.

Model EPE-Sensor    = 0.0864680055774226
Model WCE-Sensor    = -6.91744044619381
Model WDE-Sensor    = -6.91744044619381
Model Wheel-Sensor = 9.58757245842461
   wheel delta-rpm = 164.176745838329
Model ESA-Sensor    = 0.0149376166356693
   esa omega        = 5.79096473918727e-4

Real EPE-Sensor    = 0.0864680055774226
Real WCE-Sensor    = 0
Real WDE-Sensor    = -25
Real Wheel-Sensor = 34.65
  wheel delta-rpm = 139.207922699117
Real ESA Sensor    = 0.0861308576745069
```

```
  esa omega        = 4.91024576988474e-4

DONE
[32] (diagnose)
bad-sensor =M-WCE-SENSOR
suspect list =(M-WCE-B M-EPE-B)
checking suspect: M-WCE-B
checking fault: HIGH
Change M-WCE-A to -1.9174404461938.
Change M-WDE-A to -1.9174404461938.
Change M-WHEEL-A to 2.65757245842461.
Change M-ESA-A to 0.0115925373645912.
Change M-ESA-SENSOR to 0.0115925373645912.
Change M-WHEEL-SENSOR to 2.65757245842461.
Change M-WDE-SENSOR to -1.9174404461938.
Change M-WCE-SENSOR to -1.9174404461938.
disc sensor from all= M-WCE-SENSOR
checking fault: LOW
Change M-WCE-A to -11.9174404461938.
Change M-WDE-A to -11.9174404461938.
Change M-WHEEL-A to 16.5175724584246.
Change M-ESA-A to 0.0080536809460667.
Change M-ESA-SENSOR to 0.0080536809460667.
Change M-WHEEL-SENSOR to 16.5175724584246.
Change M-WDE-SENSOR to -11.9174404461938.
Change M-WCE-SENSOR to -11.9174404461938.
disc sensor from all= M-WCE-SENSOR
checking fault: ZERO
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to 0.00434790385561972.
Change M-ESA-SENSOR to 0.00434790385561972.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
disc sensor from all= M-WDE-SENSOR
checking fault: LATCHUP
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to 9.92287256169378e-4.
Change M-ESA-SENSOR to 9.92287256169378e-4.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
disc sensor from all= M-WCE-SENSOR
checking fault: LATCHDOWN
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to -0.00236332934328097.
Change M-ESA-SENSOR to -0.00236332934328097.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
disc sensor from all= M-WCE-SENSOR
suspect list =(M-EPE-B)
checking suspect: M-EPE-B
checking fault: HIGH
Change M-EPE-A to 5.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LOW
Change M-EPE-A to -4.91353199442258.
```

```
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to -0.00571894594273132.
Change M-ESA-SENSOR to -0.00571894594273132.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -4.91353199442258.
disc sensor from all= M-EPE-SENSOR
checking fault: ZERO
Change M-EPE-A to 0.
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to -0.00872440205118503.
Change M-ESA-SENSOR to -0.00872440205118503.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
Change M-EPE-SENSOR to 0.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHUP
Change M-EPE-A to 5.
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to -0.0120800186506354.
Change M-ESA-SENSOR to -0.0120800186506354.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHDOWN
Change M-EPE-A to -5.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to -0.0154356352500857.
Change M-ESA-SENSOR to -0.0154356352500857.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -5.
disc sensor from all= M-EPE-SENSOR
suspect list =()
*** Fault detected: Cause Unknown! ***
()
[33] (wce-b-on)
()
[34] (wde-a-on)
()
[35] (sp)
Change COMMAND-1 to 0.0861308576745069.
Change M-EPE-A to 0.0840637170903188.
Change M-WCE-A to -6.7250973672255.
Change M-WDE-A to -6.7250973672255.
Change M-WHEEL-A to 9.32098495097455.
Change M-ESA-A tc -0.0185352858943837.
Change M-ESA-SENSOR to -0.0185352858943837.
Change M-WHEEL-SENSOR to 9.32098495097455.
Change M-WDE-SENSOR to -6.7250973672255.
Change M-WCE-SENSOR to -6.7250973672255.
Change M-EPE-SENSOR to 0.0840637170903188.
```

```
Change R-EPE-A to ().
Change R-WCE-A to -6.91744044619381.
Change R-WDE-A to -6.91744044619381.
Change R-WHEEL-A to 9.58757245842461.
Change R-ESA-A to 0.0832203982746398.
Change R-ESA-SENSOR to 0.0832203982746398.
Change R-WHEEL-SENSOR to 9.58757245842461.
Change R-WDE-SENSOR to -6.91744044619381.
Change R-WCE-SENSOR to -6.91744044619381.
Change R-EPE-A to 0.0840637170903188.
Change R-WCE-A to -6.7250973672255.
Change R-WDE-A to -6.7250973672255.
Change R-WHEEL-A to 9.32098495097455.
Change R-ESA-A to 0.0801188557652052.
Change R-ESA-SENSOR to 0.0801188557652052.
Change R-WHEEL-SENSOR to 9.32098495097455.
Change R-WDE-SENSOR to -6.7250973672255.
Change R-WCE-SENSOR to -6.7250973672255.
Change R-EPE-SENSOR to 0.0840637170903188.

Model EPE-Sensor    = 0.0840637170903188
Model WCE-Sensor    = -6.7250973672255
Model WDE-Sensor    = -6.7250973672255
Model Wheel-Sensor = 9.32098495097455
   wheel delta-rpm = 158.022875706153
Model ESA-Sensor    = -0.0185352858943837
   esa omega        = 5.57390083794475e-4

Real EPE-Sensor    = 0.0840637170903188
Real WCE-Sensor    = -6.7250973672255
Real WDE-Sensor    = -6.7250973672255
Real Wheel-Sensor = 9.32098495097455
  wheel delta-rpm = 158.116480108517
Real ESA Sensor    = 0.0801188557652052
  esa omega        = 5.57720252230178e-4

DONE
[36] (diagnose)
bad-sensor =()
No fault found!
()
[37] (fail r-wheel-a 'zero)
ZERO
[38] (sp)
Change COMMAND-1 to 0.0801188557652052.
Change M-EPE-A to 0.0781960032268403.
Change M-WCE-A to -6.25568025814722.
Change M-WDE-A to -6.25568025814722.
Change M-WHEEL-A to 8.67037283779205.
Change M-ESA-A to -0.0218167507573545.
Change M-ESA-SENSOR to -0.0218167507573545.
Change M-WHEEL-SENSOR to 8.67037283779205.
Change M-WDE-SENSOR to -6.25568025814722.
Change M-WCE-SENSOR to -6.25568025814722.
Change M-EPE-SENSOR to 0.0781960032268403.
Change R-EPE-A to ().
Change R-EPE-A to 0.0781960032268403.
Change R-WCE-A to -6.25568025814722.
Change R-WDE-A to -6.25568025814722.
Change R-WHEEL-A to 0.
Change R-ESA-A to 0.0769231187199263.
Change R-ESA-SENSOR to 0.0769231187199263.
Change R-WHEEL-SENSOR to 0.
Change R-WDE-SENSOR to -6.25568025814722.
Change R-WCE-SENSOR to -6.25568025814722.
```

```
Change R-EPE-SENSOR to 0.0781960032268403.

Model EPE-Sensor    = 0.0781960032268403
Model WCE-Sensor    = -6.25568025814722
Model WDE-Sensor    = -6.25568025814722
Model Wheel-Sensor = 8.67037283779205
   wheel delta-rpm = 166.693248543945
Model ESA-Sensor    = -0.0218167507573545
   esa omega        = 5.87972870122025e-4

Real EPE-Sensor    = 0.0781960032268403
Real WCE-Sensor    = -6.25568025814722
Real WDE-Sensor    = -6.25568025814722
Real Wheel-Sensor = 0
  wheel delta-rpm = 158.116480108517
Real ESA Sensor    = 0.0769231187199263
  esa omega        = 5.57720252230178e-4

DONE
[39] (diagnose)
bad-sensor =M-WHEEL-SENSOR
suspect list =(M-WHEEL-A M-WDE-A M-WCE-B M-EPE-B)
checking suspect: M-WHEEL-A
checking fault: HIGH
Change M-WHEEL-A to 13.670372837792.
Change M-ESA-A to -0.0253239831953612.
Change M-ESA-SENSOR to -0.0253239831953612.
Change M-WHEEL-SENSOR to 13.670372837792.
disc sensor from all= M-WHEEL-SENSOR
checking fault: LOW
Change M-WHEEL-A to 3.67037283779205.
Change M-ESA-A to -0.0290064549990247.
Change M-ESA-SENSOR to -0.0290064549990247.
Change M-WHEEL-SENSOR to 3.67037283779205.
disc sensor from all= M-WHEEL-SENSOR
checking fault: ZERO
Change M-WHEEL-A to 0.
Change M-ESA-A to -0.0327260182761376.
Change M-ESA-SENSOR to -0.0327260182761376.
Change M-WHEEL-SENSOR to 0.
disc sensor from all= ()
suspect list =(M-WDE-A M-WCE-B M-EPE-B)
checking suspect: M-WDE-A
checking fault: HIGH
Change M-WDE-A to -1.25568025814722.
Change M-WHEEL-A to 1.74037283779205.
Change M-ESA-A to -0.0364631691378797.
Change M-ESA-SENSOR to -0.0364631691378797.
Change M-WHEEL-SENSOR to 1.74037283779205.
Change M-WDE-SENSOR to -1.25568025814722.
disc sensor from all= M-WDE-SENSOR
checking fault: LOW
Change M-WDE-A to -11.2556802581472.
Change M-WHEEL-A to 15.6003728377921.
Change M-ESA-A to -0.0403755593652785.
Change M-ESA-SENSOR to -0.0403755593652785.
Change M-WHEEL-SENSOR to 15.6003728377921.
Change M-WDE-SENSOR to -11.2556802581472.
disc sensor from all= M-WDE-SENSOR
checking fault: ZERO
Change M-WDE-A to 0.
Change M-WHEEL-A to 0..
Change M-ESA-A to -0.0444456013737052.
Change M-ESA-SENSOR to -0.0444456013737052.
Change M-WHEEL-SENSOR to 0..
```

```
Change M-WDE-SENSOR to 0.
disc sensor from all= M-WDE-SENSOR
checking fault: LATCHUP
Change M-WDE-A to 25.
Change M-WHEEL-A to -34.65.
Change M-ESA-A to -0.0481654828911352.
Change M-ESA-SENSOR to -0.0481654828911352.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25.
disc sensor from all= M-WDE-SENSOR
checking fault: LATCHDOWN
Change M-WDE-A to -25.
Change M-WHEEL-A to 34.65.
Change M-ESA-A to -0.0518853644085652.
Change M-ESA-SENSOR to -0.0518853644085652.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25.
disc sensor from all= M-WDE-SENSOR
suspect list =(M-WCE-B M-EPE-B)
checking suspect: M-WCE-B
checking fault: HIGH
Change M-WCE-A to -1.25568025814722.
Change M-WDE-A to -1.25568025814722.
Change M-WHEEL-A to 1.74037283779205.
Change M-ESA-A to -0.055972994001621.
Change M-ESA-SENSOR to -0.055972994001621.
Change M-WHEEL-SENSOR to 1.74037283779205.
Change M-WDE-SENSOR to -1.25568025814722.
Change M-WCE-SENSOR to -1.25568025814722.
disc sensor from all= M-WCE-SENSOR
checking fault: LOW
Change M-WCE-A to -11.2556802581472.
Change M-WDE-A to -11.2556802581472.
Change M-WHEEL-A to 15.6003728377921.
Change M-ESA-A to -0.0602358629603336.
Change M-ESA-SENSOR to -0.0602358629603336.
Change M-WHEEL-SENSOR to 15.6003728377921.
Change M-WDE-SENSOR to -11.2556802581472.
Change M-WCE-SENSOR to -11.2556802581472.
disc sensor from all= M-WCE-SENSOR
checking fault: ZERO
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to -0.064656383700074.
Change M-ESA-SENSOR to -0.064656383700074.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
disc sensor from all= M-WCE-SENSOR
checking fault: LATCHUP
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to -0.0687267439488177.
Change M-ESA-SENSOR to -0.0687267439488177.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
disc sensor from all= M-WCE-SENSOR
checking fault: LATCHDOWN
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to -0.0727971041975614.
```

```
Change M-ESA-SENSOR to -0.0727971041975614.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
disc sensor from all= M-WCE-SENSOR
suspect list =(M-EPE-B)
checking suspect: M-EPE-B
checking fault: HIGH
Change M-EPE-A to 5.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LOW
Change M-EPE-A to -4.92180399677316.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to -0.0768674644463052.
Change M-ESA-SENSOR to -0.0768674644463052.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -4.92180399677316.
disc sensor from all= M-EPE-SENSOR
checking fault: ZERO
Change M-EPE-A to 0.
Change M-WCE-A to 0.
Change M-WDE-A to 0..
Change M-WHEEL-A to 0..
Change M-ESA-A to -0.0805876642040523.
Change M-ESA-SENSOR to -0.0805876642040523.
Change M-WHEEL-SENSOR to 0..
Change M-WDE-SENSOR to 0..
Change M-WCE-SENSOR to 0.
Change M-EPE-SENSOR to 0.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHUP
Change M-EPE-A to 5.
Change M-WCE-A to -25.
Change M-WDE-A to -25..
Change M-WHEEL-A to 34.65.
Change M-ESA-A to -0.084658024452796.
Change M-ESA-SENSOR to -0.084658024452796.
Change M-WHEEL-SENSOR to 34.65.
Change M-WDE-SENSOR to -25..
Change M-WCE-SENSOR to -25.
Change M-EPE-SENSOR to 5.
disc sensor from all= M-EPE-SENSOR
checking fault: LATCHDOWN
Change M-EPE-A to -5.
Change M-WCE-A to 25.
Change M-WDE-A to 25..
Change M-WHEEL-A to -34.65.
Change M-ESA-A to -0.0887283847015398.
Change M-ESA-SENSOR to -0.0887283847015398.
Change M-WHEEL-SENSOR to -34.65.
Change M-WDE-SENSOR to 25..
Change M-WCE-SENSOR to 25.
Change M-EPE-SENSOR to -5.
disc sensor from all= M-EPE-SENSOR
suspect list =()
The culprit is: M-WHEEL-A
()
[40] (transcript-off)
```

Appendix L: Program Run - Combined Model

This listing is a diagnostic run, using filters in the real-world model, to ensure an accurate simulation of the Satellite's pitch control channel, but using no filters in the computer-world model, to ensure operation of the model-based reasoner. The (sp) function had to be used several times between changes to the system to allow the state-variables in those objects with filters to settle down.
To shorten this listing, the sensor readouts for most of these (sp) invocations has been edited out.

```
[3] (reset-system)
OK
[4] (load "mpchl2.s")
OK
[5] (set-pitch .1)

Model EPE-Sensor    = 0.0976
Model WCE-Sensor    = -1.7324
Model WDE-Sensor    = -1.7324
Model Wheel-Sensor = 2.4011064
   wheel delta-rpm = 2.4011064
Model ESA-Sensor    = 0.0999757352786159
   esa omega        = 8.46936173965186e-6

Real EPE-Sensor    = 0.0976
Real WCE-Sensor    = -5.5991168
Real WDE-Sensor    = -3.8113110139316
Real Wheel-Sensor = 5.2824770653092
  wheel delta-rpm = 9.23523973535579
Real ESA Sensor    = 0.0998267815666702
  esa omega        = 3.25752270166515e-5

DONE
[6] (sp)
[7] (sp)
[8] (sp)
[10] (sp)
[11] (sp)

Model EPE-Sensor    = 0.091316340712002
Model WCE-Sensor    = -1.62086504763804
Model WDE-Sensor    = -1.62086504763804
Model Wheel-Sensor = 2.24651895602632
   wheel delta-rpm = 14.0074529717373
Model ESA-Sensor    = 0.0999757352786159
   esa omega        = 4.94081337956554e-5

Real EPE-Sensor    = 0.091316340712002
Real WCE-Sensor    = -1.53836534943618
Real WDE-Sensor    = -1.76473621808494
Real Wheel-Sensor = 2.44592439826572
  wheel delta-rpm = 52.4578350883391
Real ESA Sensor    = 0.0915409842180378
  esa omega        = 1.85033191966064e-4

DONE
```

```
[12] (diagnose)
No fault found!
()
[13] (fail r-epe-a 'zero)
ZERO
[14] (sp)

Model EPE-Sensor    = 0.0893440005968049
Model WCE-Sensor    = -1.58585601059329
Model WDE-Sensor    = -1.58585601059329
Model Wheel-Sensor = 2.1979964306823
   wheel delta-rpm = 16.2054494024196
Model ESA-Sensor    = 0.0988344120002407
   esa omega        = 5.71610708891187e-5

Real EPE-Sensor    = 0
Real WCE-Sensor    = 3.68604747023245
Real WDE-Sensor    = 1.84827374792807
Real Wheel-Sensor = -2.5617074146283
  wheel delta-rpm = 48.6027629888543
Real ESA Sensor    = 0.0894856023767895
  esa omega        = 1.71435293870847e-4

DONE
[15] (sp)
[16] (sp)
[17] (sp)
[18] (sp)
[19] (sp)
[20] (sp)
[21] (sp)
[22] (sp)

Model EPE-Sensor    = 0.0767927842732255
Model WCE-Sensor    = -1.36307192084975
Model WDE-Sensor    = -1.36307192084975
Model Wheel-Sensor = 1.88921768229776
   wheel delta-rpm = 32.3068079159412
Model ESA-Sensor    = 0.0948812845400393
   esa omega        = 1.13954984624403e-4

Real EPE-Sensor    = 0
Real WCE-Sensor    = 0.0106087610796947
Real WDE-Sensor    = 0.104650335238562
Real Wheel-Sensor = -0.145045364640647
  wheel delta-rpm = 33.7220394670155
Real ESA Sensor    = 0.0773118845446167
  esa omega        = 1.18946895000143e-4

DONE
[23] (diagnose)
The culprit is: M-EPE-A
()
[24] (epe-b-on)
()
[25] (sp)

Model EPE-Sensor    = 0.0754563993155459
Model WCE-Sensor    = -1.33935108785094
Model WDE-Sensor    = -1.33935108785094
Model Wheel-Sensor = 1.8563406077614
   wheel delta-rpm = 34.1631485237026
Model ESA-Sensor    = 0.091550355797O732
   esa omega        = 1.20502807794228e-4
```

```
Real EPE-Sensor    = 0.0754563993155459
Real WCE-Sensor    = -3.42201469500399
Real WDE-Sensor    = -2.11524220464202
Real Wheel-Sensor = 2.93172569563384
  wheel delta-rpm = 36.6537651626494
Real ESA Sensor    = 0.0766006918663077
  esa omega        = 1.29287896730743e-4

DONE
[26] (sp)
[27] (sp)
[28] (sp)
[29] (sp)
[30] (sp)

Model EPE-Sensor    = 0.0666999694085924
Model WCE-Sensor    = -1.18392445700252
Model WDE-Sensor    = -1.18392445700252
Model Wheel-Sensor = 1.64091929740549
   wheel delta-rpm = 42.8953018514489
Model ESA-Sensor    = 0.0884970698450377
   esa omega        = 1.51303510877935e-4

Real EPE-Sensor    = 0.0666999694085924
Real WCE-Sensor    = -1.15604167655712
Real WDE-Sensor    = -1.36542729232223
Real Wheel-Sensor = 1.89248222715862
  wheel delta-rpm = 66.5136201350415
Real ESA Sensor    = 0.0657298759526331
  esa omega        = 2.34611806264586e-4

DONE
[31] (diagnose)
No fault found!
()
[32] (fail r-wce-a 'high)
HIGH
[33] (sp)

Model EPE-Sensor    = 0.0641523589297699
Model WCE-Sensor    = -1.13870437100342
Model WDE-Sensor    = -1.13870437100342
Model Wheel-Sensor = 1.57824425821073
   wheel delta-rpm = 44.4735461096596
Model ESA-Sensor    = 0.0867637649818772
   esa omega        = 1.56870411843388e-4

Real EPE-Sensor    = 0.0641523589297699
Real WCE-Sensor    = 4.0256855305017
Real WDE-Sensor    = 2.25916334898757
Real Wheel-Sensor = -3.13120040169677
  wheel delta-rpm = 60.7275733668478
Real ESA Sensor    = 0.0631533542225872
  esa omega        = 2.14202830168243e-4

DONE
[34] (sp)
[35] (sp)
[36] (sp)
[37] (sp)
[38] (sp)
[39] (sp)
[40] (sp)
[41] (sp)
```

```
Model EPE-Sensor    = 0.0534192969715531
Model WCE-Sensor    = -0.955285030704803
Model WDE-Sensor    = -0.955285030704803
Model Wheel-Sensor = 1.32402505255686
   wheel delta-rpm = 52.9058839091103
Model ESA-Sensor    = 0.0808456172562096
   esa omega        = 1.86613583214089e-4

Real EPE-Sensor    = 0.0534436838625508
Real WCE-Sensor    = 4.11736175968981
Real WDE-Sensor    = 3.95346468920179
Real Wheel-Sensor = -5.47950205923369
  wheel delta-rpm = -16.9892669664936
Real ESA Sensor    = 0.0552232898506788
  esa omega        = -5.99258107140743e-5

DONE
[42] (diagnose)
The culprit is: M-WCE-A
()
[43] (fix m-wce-a)
ON
[44] (sp)

Model EPE-Sensor    = 0.0538979308942625
Model WCE-Sensor    = -0.95668827337316
Model WDE-Sensor    = -0.95668827337316
Model Wheel-Sensor = 1.3259699468952
   wheel delta-rpm = 48.6258789085623
Model ESA-Sensor    = 0.0736523720283422
   esa omega        = 1.71516830068473e-4

Real EPE-Sensor    = 0.0538979308942625
Real WCE-Sensor    = 4.06354241344053
Real WDE-Sensor    = 3.95869664997481
Real Wheel-Sensor = -5.48675355686508
  wheel delta-rpm = -27.9689655260249
Real ESA Sensor    = 0.0561320160130899
  esa omega        = -9.86542231213732e-5

DONE
[45] (sp)
[46] (sp)
[47] (sp)
[48] (sp)
[49] (sp)
[50] (sp)
[51] (sp)
[52] (sp)
[53] (sp)

Model EPE-Sensor    = 0.0771126374075432
Model WCE-Sensor    = -1.36874931398389
Model WDE-Sensor    = -1.36874931398389
Model Wheel-Sensor = 1.89708654918168
   wheel delta-rpm = 62.8001143700916
Model ESA-Sensor    = 0.0626321111101381
   esa omega        = 2.21513251512642e-4

Real EPE-Sensor    = 0.0771126374075432
Real WCE-Sensor    = 3.23164734531692
Real WDE-Sensor    = 3.36296137432913
Real Wheel-Sensor = -4.66106446482018
  wheel delta-rpm = -121.086808896688
Real ESA Sensor    = 0.083713706512694
```

```
  esa omega        = -4.271064316209l8e-4

DONE
[54] (diagnose)
The culprit is: M-WCE-A
()
[55] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-A
   STATUS : HIGH
   STATE : 3.23164734531692
   INPUT-LIST : (R-EPE-B)
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0.0771126374075432
   YN-1 : 0.0996255016722862
   YN-2 : 0.0921836552187563
   LIMIT : 25
   TOLERANCE : 1.e-6
()
[56] (fix r-wce-a)
ON
[57] (describe r-wce-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WCE

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WCE-A
   STATUS : ON
   STATE : 3.23164734531692
   INPUT-LIST : (R-EPE-B)
   OUTPUT-LIST : (R-WDE-A R-WCE-SENSOR)
   GAIN : -17.75
   XN-1 : 0.0771126374075432
   YN-1 : 0.0996255016722862
   YN-2 : 0.0921836552187563
   LIMIT : 25
   TOLERANCE : 1.e-6
()
[58] (sp)

Model EPE-Sensor    = 0.0817045775563893
Model WCE-Sensor    = -1.45025625162591
Model WDE-Sensor    = -1.45025625162591
Model Wheel-Sensor = 2.01005516475351
   wheel delta-rpm = 59.7772560840267
Model ESA-Sensor    = 0.0529874504023705
   esa omega        = 2.10850800105913e-4

Real EPE-Sensor    = 0.0817045775563893
```

```
Real WCE-Sensor    = -1.8934141177262
Real WDE-Sensor    = -0.202217687191201
Real Wheel-Sensor = 0.280273714447005
  wheel delta-rpm = -121.001586292193
Real ESA Sensor    = 0.0886114272232782
  esa omega        = -4.26805828088368e-4

DONE
[59] (sp)
[60] (sp)
[61] (sp)
[62] (sp)
[63] (sp)
[64] (sp)
[65] (sp)
[66] (sp)

Model EPE-Sensor    = 0.116761393576748
Model WCE-Sensor    = -2.07251473598728
Model WDE-Sensor    = -2.07251473598728
Model Wheel-Sensor = 2.87250542407837
   wheel delta-rpm = 79.8917980009012
Model ESA-Sensor    = 0.0417864964063049
   esa omega        = 2.81800313930623e-4

Real EPE-Sensor    = 0.116761393576748
Real WCE-Sensor    = -2.52332649242971
Real WDE-Sensor    = -2.26174401031693
Real Wheel-Sensor = 3.13477719829926
  wheel delta-rpm = -86.3907738004784
Real ESA Sensor    = 0.123250144995333
  esa omega        = -3.04723986527498e-4

DONE
[67] (diagnose)
No fault found!
()
[68] (fail r-wde-a 'latchdown)
LATCHDOWN
[69] (sp)

Model EPE-Sensor    = 0.120292141515445
Model WCE-Sensor    = -2.13518551189915
Model WDE-Sensor    = -2.13518551189915
Model Wheel-Sensor = 2.95936711949222
   wheel delta-rpm = 82.8511651203934
Model ESA-Sensor    = 0.0401418743031938
   esa omega        = 2.92238814554923e-4

Real EPE-Sensor    = 0.120292141515445
Real WCE-Sensor    = -2.56284574904082
Real WDE-Sensor    = -25
Real Wheel-Sensor = 34.65
  wheel delta-rpm = -51.7407738004784
Real ESA Sensor    = 0.124646052947139
  esa omega        = -1.82503919862007e-4

DONE
[70] (sp)
[71] (sp)
[72] (sp)
[73] (sp)

Model EPE-Sensor    = 0.121654547676407
Model WCE-Sensor    = -2.15936822125623
```

```
Model WDE-Sensor    = -2.15936822125623
Model Wheel-Sensor = 2.99288435466114
   wheel delta-rpm = 85.8440494750545
Model ESA-Sensor    = 0.0384371008783302
   esa omega        = 3.02795539673214e-4

Real EPE-Sensor    = 0.121654547676407
Real WCE-Sensor    = -2.31983291060259
Real WDE-Sensor    = -25
Real Wheel-Sensor = 34.65
  wheel delta-rpm = -51.7407738004784
Real ESA Sensor    = 0.124646052947139
  esa omega        = -1.82503919862007e-4

DONE
[74] (diagnose)
The culprit is: M-WDE-A
()
[75] (wde-b-on)
()
[76] (sp)

Model EPE-Sensor    = 0.121654547676407
Model WCE-Sensor    = -2.15936822125623
Model WDE-Sensor    = -2.15936822125623
Model Wheel-Sensor = 2.99288435466114
   wheel delta-rpm = 66.1584712483602
Model ESA-Sensor    = 0.0137309314130346
   esa omega        = 2.33359098599179e-4

Real EPE-Sensor    = 0.121654547676407
Real WCE-Sensor    = -2.27264222606203
Real WDE-Sensor    = -1.46143815096253
Real Wheel-Sensor = 2.02555327723407
  wheel delta-rpm = -49.7152205232443
Real ESA Sensor    = 0.125671330891928
  esa omega        = -1.75359237132482e-4

DONE
[77] (sp)
[78] (sp)
[79] (sp)
[80] (sp)

Model EPE-Sensor    = 0.127706123768205
Model WCE-Sensor    = -2.26678369688565
Model WDE-Sensor    = -2.26678369688565
Model Wheel-Sensor = 3.14176220388351
   wheel delta-rpm = 78.4865994933728
Model ESA-Sensor    = 0.00788819369912571
   esa omega        = 2.76843792855057e-4

Real EPE-Sensor    = 0.127706123768205
Real WCE-Sensor    = -2.43164314838878
Real WDE-Sensor    = -2.12868853269769
Real Wheel-Sensor = 2.950362306319
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.1321407509155
  esa omega        = -1.02622717737412e-4

DONE
[81] (diagnose)
No fault found!
()
[82] (fail r-wheel-a 'zero)
```

```
ZERO
[83] (sp)

Model EPE-Sensor    = 0.128969372893528
Model WCE-Sensor    = -2.28920636886012
Model WDE-Sensor    = -2.28920636886012
Model Wheel-Sensor = 3.17284002724012
   wheel delta-rpm = 81.6594395206129
Model ESA-Sensor    = 0.00626981518102173
   esa omega        = 2.88035271055582e-4

Real EPE-Sensor    = 0.128969372893528
Real WCE-Sensor    = -2.44953520764726
Real WDE-Sensor    = -2.21497355456307
Real Wheel-Sensor = 0
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.13331680726077
  esa omega        = -1.02622717737412e-4

DONE
[84] (sp)
[85] (sp)

Model EPE-Sensor    = 0.131265034879496
Model WCE-Sensor    = -2.32995436911106
Model WDE-Sensor    = -2.32995436911106
Model Wheel-Sensor = 3.22931675558793
   wheel delta-rpm = 88.0898346676149
Model ESA-Sensor    = 0.00283924980056922
   esa omega        = 3.10717040855064e-4

Real EPE-Sensor    = 0.131265034879496
Real WCE-Sensor    = -2.46785898030333
Real WDE-Sensor    = -2.32832205452975
Real Wheel-Sensor = 0
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.135668919951312
  esa omega        = -1.02622717737412e-4

DONE
[86] (diagnose)
The suspects are: (M-WHEEL-SENSOR M-WHEEL-A)
()
[87] (wheel-b-on)
()
[88] (describe r-wheel-a)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WHEEL

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WHEEL-A
   STATUS : OFF
   STATE : 0
   INPUT-LIST : (R-WDE-B)
   OUTPUT-LIST : (R-ESA-A R-WHEEL-SENSOR)
   GAIN : -1.386
   DELTA-RPM : -29.0940535921449
   LIMIT : 35
   TOLERANCE : 1.e-6
```

```
()
[89] (describe r-wheel-b)

  INSTANCE DESCRIPTION
  ====================

 Instance of Class WHEEL

 Class Variables :

Instance Variables :
   FAULT-LIST : (HIGH LOW ZERO LATCHUP LATCHDOWN)
   NAME : R-WHEEL-A
   STATUS : ON
   STATE : 0
   INPUT-LIST : (R-WDE-B)
   OUTPUT-LIST : (R-ESA-A R-WHEEL-SENSOR)
   GAIN : -1.386
   DELTA-RPM : 0
   LIMIT : 35
   TOLERANCE : 1.e-6
()
[90] (sp)

Model EPE-Sensor    = 0.13241286587248
Model WCE-Sensor    = -2.35032836923653
Model WDE-Sensor    = -2.35032836923653
Model Wheel-Sensor = 3.25755511976183
   wheel delta-rpm = 69.5864184455545
Model ESA-Sensor    = -0.030431679085332
   esa omega        = 2.68331388585401e-4

Real EPE-Sensor    = 0.13241286587248
Real WCE-Sensor    = -2.4824486232592
Real WDE-Sensor    = -2.37177202458425
Real Wheel-Sensor = 3.28727602607377
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.136713490362075
  esa omega        = -7.95946778258097e-5

DONE
[91] (sp)
[92] (sp)

Model EPE-Sensor    = 0.133432366593385
Model WCE-Sensor    = -2.38191082663296
Model WDE-Sensor    = -2.38191082663296
Model Wheel-Sensor = 3.30132840571329
   wheel delta-rpm = 69.5864184455545
Model ESA-Sensor    = -0.0336396382836957
   esa omega        = 2.91554840631963e-4

Real EPE-Sensor    = 0.134192159246927
Real WCE-Sensor    = -2.49433098607845
Real WDE-Sensor    = -2.42766672425584
Real Wheel-Sensor = 3.36474607981859
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.13800072356495
  esa omega        = -3.25469483898684e-5

DONE
[93] (diagnose)
No fault found!
()
[94] (fail r-wde-b 'latchup)
```

```
LATCHUP
[95] (sp)

Model EPE-Sensor    = 0.134688706199391
Model WCE-Sensor    = -2.38191082663296
Model WDE-Sensor    = -2.38191082663296
Model Wheel-Sensor = 3.30132840571329
   wheel delta-rpm = 69.5864184455545
Model ESA-Sensor    = -0.0336396382836957
   esa omega        = 2.91554840631963e-4

Real EPE-Sensor    = 0.134688706199391
Real WCE-Sensor    = -2.48252212871888
Real WDE-Sensor    = 25
Real Wheel-Sensor = -34.65
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.13853737807022
  esa omega        = -1.54767015055359e-4

DONE
[96] (sp)
[97] (sp)
[98] (sp)
[99] (sp)
[100] (sp)

Model EPE-Sensor    = 0.134688706199391
Model WCE-Sensor    = -2.40002153768849
Model WDE-Sensor    = -2.40002153768849
Model Wheel-Sensor = 3.32642985123625
   wheel delta-rpm = 69.5864184455545
Model ESA-Sensor    = -0.0353438632293185
   esa omega        = 3.03288072325359e-4

Real EPE-Sensor    = 0.135212480996535
Real WCE-Sensor    = -2.42718322309917
Real WDE-Sensor    = 25
Real Wheel-Sensor = -34.65
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.13853737807022
  esa omega        = -1.54767015055359e-4

DONE
[101] (diagnose)
The culprit is: M-WDE-B
()
[102] (fix r-wde-b)
ON
[103] (sp)

Model EPE-Sensor    = 0.135212480996535
Model WCE-Sensor    = -2.40002153768849
Model WDE-Sensor    = -2.40002153768849
Model Wheel-Sensor = 3.32642985123625
   wheel delta-rpm = 69.5864184455545
Model ESA-Sensor    = -0.0512407472822511
   esa omega        = 1.17514097461358e-4

Real EPE-Sensor    = 0.135212480996535
Real WCE-Sensor    = -2.41893897668886
Real WDE-Sensor    = -2.38806092695466
Real Wheel-Sensor = 3.30985244475916
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.139390744883019
  esa omega        = -1.43092256427613e-4
```

```
DONE
[104] (sp)
[105] (sp)

Model EPE-Sensor    = 0.137514912085292
Model WCE-Sensor    = -2.44088968951393
Model WDE-Sensor    = -2.44088968951393
Model Wheel-Sensor = 3.38307310966631
   wheel delta-rpm = 69.5864184455545
Model ESA-Sensor    = -0.0525876659062378
   esa omega        = 1.41252631900116e-4

Real EPE-Sensor    = 0.137514912085292
Real WCE-Sensor    = -2.51668571521074
Real WDE-Sensor    = -2.46693012192406
Real Wheel-Sensor = 3.41916514898675
  wheel delta-rpm = -29.0940535921449
Real ESA Sensor    = 0.14213092373858
  esa omega        = -9.57166577999724e-5

DONE
[106] (diagnose)
No fault found!
()
[107] (transcript-off)
```

Appendix M: Research Paper Summary

Chronological Order

84Davis	Constraint propagation and constraint suspension
85Fink	Uses rule and models (first principles) to diagnose faults. Similar to Skinner's BDS.
85Scarl	Model-based reasoning using symbolic inversion techniques on Space Shuttle Oxygen loading system.
86Adams	Applies to Space vehicle electrical system. Includes recovery
87Blasdel	Describes PARAGON (generic model-based development tool) application to a satellites EPS.
87Dekleer	Constraint propagation/suspension & truth maintenance
87Golden	Discussion of space support diagnostic systems: PARAGON (model-based) a general purpose development tool; STARPLAN to automate the ground control (combines rules, frames and PARAGON representations).
87Hamscher	Discusses issues in model-based troubleshooting. Good basic overview of MBR. Good examples - again deKleer's work.
87Scarl	Uses model-based reasoning for sensor validation. Sensors treated like any other component.
88Ben-bassat	Example of troubleshooting a dot-matrix printer. Builds the model using graphic tools and a universal knowledge-base of common components and their function.
88Chu	Introduces a generic expert system shell for diagnostic reasoning. Domain knowledge is represented as 5 different classes of object. Not model-based reasoning. Reasoning follows human's method. Rules are replaced by object!
88Davis	Good overview of model-based reasoning. Updates research from Davis84. Gives overview of other systems including DeKleer's GDE.
88Day	Independent work done in UK on "deep models" as opposed to rules. For complex systems.....written LIST, recast in C.
88Hansen	Diagnoses multiple faults using bi-directional simulation (constraint propagation) and models which assume both normal functioning and malfunctioning.

88Hofmann Diagnoses analog circuits using a generic schematic knowledge representation, acquisition, and manipulation system (SKRAM). SKRAM accepts knowledge and constructs models. Uses fault models to constrain the search.

88Khaksari Uses shallow and deep knowledge in Expert Diagnostic System (EDS). Looks very similar to Skinner's BDS. Uses directed graph techniques.

88Skinner Applies both rules and models (BDS) to INS system

88Sticklen Good overall discussion on "functional approach" to diagnosis. Gives model representation of a clothes pin. Medical Apps.

89Chen Uses candidate ordering to increase computing efficiency

89Dekleer Similar to Struss, uses behavioral models and probabilistic info to derive additional diagnostic discrimination. For efficiency.

89Dvorak Uses systems dynamic behavior to diagnose continuously changing systems.

89Gallanti Uses quantative/qualitative models & different levels of abstraction to diagnose complex systems

89Hamscher Uses temporally coarse representation of behavior for model-based troubleshooting of complex digital circuits. For efficiency.

89Padalkar Uses hierarchial Graph-based techniques for Real-time diagnoses. Very fast - suitable for automatic space systems.

89Resnick Discusses performance issues in model-based reasoning. Systems that learn are discussed.

89Struss Builds on deKleer's GDE using device models and knowledge about how a components behave when they are faulty. For efficiency.

89Tonga Work by deKleer and Williams has been extended to diagnose steady state faults in analog feedback systems. (still requires a tech)

89Tongb Builds on work by deKleer. Uses model-based reasoning to automatically generate n-ary trees used for diagnosis.

89Yost Refines Skinners work - uses model behavior (function) as well as connectivity.

90Fulton AI expert article on model-based reasoning. Discusses work by Scarl.

Bibliography

Adams, Thomas L. "Model-Based Reasoning for Automated Fault Diagnosis and Recovery Planning in Space Power Systems," *Proceedings of the 21st Intersociety Energy Conversion Engineering Conference.* 1763-1767. San Diego, California, August 1986.

Barry, John M. "Applications of Expert Systems to Command and Control of Satellites," *Proceedings of the Third Annual Conference on Aerospace Applications of Artificial Intelligence (AAAIC 87).* 276-282. Dayton, Ohio, October 1987.

Ben-Bassat, Moshe, et al. "AI-TEST: A Real Life Expert System for Electronic Troubleshooting (A Description and Case Study)," *Proceedings of the Fourth Conference on Artifical Intelligence Applications.* 2-10. San Diego, California, March 1988.

Blasdel, Arthur N. "Automated Fault Handling of a Satellite Electrical Power Subsystem using A Model-Based Expert System," *Proceedings of the 22nd Intersociety Energy Conversion Engineering Conference*, 601-606. Philadelphia, Pennsylvania, August 1987.

Bourret, P. and J.A. Reggia. "A Method for Interactive Satellite Failure Diagnosis: Towards a Connectionist Solution," *Proceedings of the 1989 Goddard Conference on Space Applications of Artificial Intelligence*, 143-152. Goddard Space Flight Center, April 1989.

Brown. J.S., R. Burton and J. de Kleer. "Pedagogical and Knowledge Engineering Techniques in SOPHIE I, II and III," in: D.H. Sleeman and J.S. Brown (Eds.), *Intelligent Tutoring Systems.* Academic Press, New York, 1982.

Busse, Carl. "An Expert System for Satellite and Instrument Data Anomaly and Fault Isolation," *Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications.* 356-367. University of Alabama, Huntsville, Alabama, November 1988.

Chen, Jiah-shing and Sargur N. Srihari. "Candidate Ordering and Elimination in Model-Based Fault Diagnosis," *Proceedings of the Eleventh Internatianal Joint Conference on Artificial Intellegence.* 1363-1368. Detroit, Michigan, August 89.

Chu, Wei-Han. "Generic Expert System Shell for Diagnostic Reasoning," *Proceedings of the First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-88).* 7-12. University of Tennessee Space Institute (UTSI), Tullahoma, Tennessee, June 1988.

Cochran, Lt Col Curtis D. et al. *Space Handbook* (Twelfth Revision). Maxwell Air Force Base, Alabama: Air University Press, 1985.

Davis, Randall and Walter C. Hamscher. *Model-Based Reasoning: Troubleshooting.* Report: A.I. Memo No. 1059. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, July 1988 (AD-A201614).

Davis, Randall. "Diagnostic Reasoning Based on Structure and Behavior," Artificial," *Artificial Intelligence, 24(3)*: 347-410 (December 1984).

Day, P.O. and M.K. Hook. "An AI-Based Fault Diagnosis Aid for Complex Electronic Systems," *AIAA/IEEE 8th Digital Avionics Systems Conference.* 17-20. San Jose, California, October 1988.

de Kleer, Johan and Brian C. Williams. "Diagnosis with Behavioral Modes," *Proceedings of the Eleventh Internatianal Joint Conference on Artificial Intellegence.* 1324-1330. Detroit, Michigan, August 89.

de Kleer, Johan and Brian C. Williams. "Diagnosing Multiple Faults," *Artificial Intellegence, 32(1)*: 97-130 (April 1987).

de Kleer, J. and J.S. Brown. *Assumptions and Ambiguities in Mechanistic Mental Models.* Report: CIS-9. Xerox PARC, Palo Alto, CA, 1982.

de Kleer, Johan. "The Origin and Resolution of Ambiguities in Causal Arguments," *Proceedings of the Sixth International Conference on Artificial Intelligence.* 197-203. Tokyo, Japan, August 1979.

de Kleer, Johan. *Local Methods for Localizing Faults in Electronic Circuits.* Report: A.I. Memo No. 394. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1976.

Dietz, W.E., E.L. Kiech, and M. Ali. "Pattern-Based Fault Diagnosis Using Neural Networks," *Proceedings of the First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-88).* 13-23. University of Tennessee Space Institute (UTSI), Tullahoma, Tennessee, June 1988.

Dvorak, Daniel and Benjamin Kuipers. "Model-Based Monitoring of Dynamic Systems," *Proceedings of the Eleventh Internatianal Joint Conference on Artificial Intellegence.* 1238-1343. Detroit, Michigan, August 89.

Elliott, Captain Grady N. *Decision-Analytic Approach to Rule-Based Expert System Development using GPS as the Model.* MS thesis, AFIT/GSO/ENS/89D-5. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

Fink, Pamela K., John C. Lusth and Joe W. Duran. "The Integrated Diagnostic Model (IDM) - Troubleshooter and Theoretician," *Proceedings of the Automatic Testing Conference (AUTOTESTCON 85).* 63-68. Uniondale, NY, October 1985.

Fogiel, Dr. M. *The Essentials of Automatic Control Systems/Robotics I.* Research and Education Association, Piscataway, New Jersey, 1987.

Fulton, Steven, L. and Charles O. Pepe. "An Introduction to Model-Based Reasoning," *AI Expert, 5*: 48-55 (January 1990).

Gallanti, Massimo et al. "A Diagnostic Algorithm based on Models at Different Level of Abstraction," *Proceedings of the Eleventh Internatianal Joint Conference on Artificial Intellegence.* 1350-1355. Detroit, Michigan, August 89.

Genesereth, M.R. "The Use of Design Descriptions in Automated Diagnosis," *Artificial Intelligence, 24(3)*: 411-436 (December 1984).

Genesereth, M.R. *The Use of Hierarchical Models in the Automated Diagnosis of Computer Systems.* Report: Stanford HPP Memo No 81-20. Stanford, CA, 1981.

Giarratano, Joseph and Gary Riley. *Expert Systems: Principles and Programming.* Boston: PWS-KENT Publishing Company, 1989.

Gilmore, John F. and Kurt Gingher. "A Survey of Diagnostic Expert Systems," *Proceedings of Applications of Artificial Intelligence V (SPIE), 786*: 2-11. Orlando, Florida, May 1987.

Golden, Constance, J. "AI Application for Space Support and Satellite Autonomy," *Proceedings of the Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program.* AIAA-87-1682. Arlington, VA, March 1987.

Grun, Rainer and Leliner Albrecht. "Ground Based Operations Support By Artificial Intelligence," *Proceedings of the 25th Space Congress.* 6-1 to 6-10. Cocoa Beach, Florida, April 1988.

Hamscher, Walter. "Temporally Coarse Representation of Behavior for Model-based Troubleshooting of Digital Circuits," *Proceedings of the Eleventh Internatianal Joint Conference on Artificial Intellegence.* 887-893. Detroit, Michigan, August 89.

Hamscher, Walter and Davis, Randall. *Issues in Model-Based Troubleshooting.* Report: A.I. Memo No. 893. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, March 1987 (AD-A183617).

Hansen, Tim. "Diagnosing Multiple Faults Using Knowledge about Malfunctioning Behavior," *Proceedings of the First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-88).* 29-36. University of Tennessee Space Institute (UTSI), Tullahoma, Tennessee, June 1988.

Hester, Lieutenant Gina L. *A Prototype Fault Diagnosis System for NASA Space Station Power Management and Control.* MS thesis. Naval Postgraduate School, Monterey, California, September 1988 (AD-A202032).

Hofmann, G., G. Collins, and A. Broderson. "A Paradigm for Building Diagnostic Expert Systems by Specializing Generic Device and Reasoning Models," *Proceedings of the First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-88).* 37-42. University of Tennessee Space Institute (UTSI), Tullahoma, Tennessee, June 1988.

Howlin, Katherine et al. "MOORE: A Prototype Expert System for Diagnosing Spacecraft Problems," *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence.* 175-190. Goddard Space Flight Center, August 1988.

Khaksari, Gholam, H. "Expert Diagnostic System," *Proceedings of the First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-88).* 43-53. University of Tennessee Space Institute (UTSI), Tullahoma, Tennessee, June 1988.

Koons, H.C. and D.J. Gorney. *Spacecraft Environmental Anomalies Expert System: A Status Report.* Report ATR-88(9562)-1. Laboratory Operations, The Aerospace Corporation, El Segundo, California, December 1988.

Longoni, F., P. Donzelli and A. Biotti. "Artificial Intelligence Application for Satellite Control: CANDIES, An Expert System For Spacecraft Diagnosis," *Proceeding of the Tenth Triennial World Congress of IFAC.* Volume 6. 293-297. Munich, Federal Republic of Germany, July 1987.

Marsh, Christopher A. "The ISA Expert System: A Prototype System for Failure Diagnosis on the Space Station," *Proceedings of the First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-88).* 60-74. University of Tennessee Space Institute (UTSI), Tullahoma, Tennessee, June 1988.

Murugesan, S. "Considerations in Development of Expert Systems for Real-Time Space Applications," *Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications.* 487-496. University of Alabama, Huntsville, Alabama, November 1988.

Norman, Capt Douglas, O. *Reasoning in Real Time for the Pilot Associate: An Examination of a Model-Based Approach to Reasoning in Real-Time for Artificial Intellegence Systems Using Distributed Architecture.* MS thesis, AFIT/GCS/ENG/85D-12. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985 (AD-A163947).

OOH, Department of the Air Force. *Orbital Operations Handbook.* Volume 1, Part 1, Document No. 26700-270-002-02. Headquarters Space and Missile Systems Organization, 17 August 1977.

Padalkar, S., G. Karsai and J. Sztipanovits. "Graph-Based Real-Time Fault Diagnosis," *Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications.* 115-123. University of Alabama, Huntsville, Alabama, November 1988.

Passani, Michael J. and Anne F. Brindle. "Automated Diagnosis of Attidude Control Anomalies," *Proceedings of the Annual Rocky Mountain Guidance and Control Conference.* 255-262. Keystone, Colorado, February 1986.

Rampino, Capt Michael A. *NAVARES: A Prototype Expert System for NAVSTAR Anomaly Resolution.* MS Thesis, AFIT/GSO/ENS/87D-10. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A189491).

Resnick, Paul. *Generalizing on Multiple Grounds: Performance Learning in Model-Based Troubleshooting.* Report: AI-TR 1052. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 1989 (AD-A207960).

Scarl, Ethan, A. et al. "Diagnosis and Sensor Validation through Knowledge of Structure and Function," *IEEE Transactions on Systems, Man, and Cybernetics, 17(3)*: 360-368 (May/June 1987).

Scarl, Ethan, A. et al. "A Fault Detection and Isolation Method Applied to Liquid Oxygen Loading for the Space Shuttle," *Proceedings of the 9th International Joint Conference on Artificial Intelligence.* 414-416. 1985.

Seifert, Capt Mark W. *Satellite Anomaly Detection and Identification using Expert Systems and Neural Processing.* MS thesis, AFIT/GCE/ENG/89D-7. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

Skinner, Capt James M. *A Diagnositic System Blending Deep and Shallow Reasoning.* MS thesis, AFIT/GCE/ENG/88D-5. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988 (AD-A202547).

Sticklen, Jon, W.W. Bond and D.C. St.Clair. "Functional Reasoning in Diagnosic Problem Solving," *Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications.* 29-38. University of Alabama, Huntsville, Alabama, November 1988.

Struss, Peter and Oskar Dressler. "Physical Negation - Integrating Fault Models into the General Diagnostic Engine," *Proceedings of the Eleventh Internatianal Joint Conference on Artificial Intellegence.* 1318-1323. Detroit, Michigan, August 89.

Sussman, G.J. and G. Steele. "Constraints - a Language for Expressing Almost-Hierarchical Descriptions," *Artificial Intelligence, 14:* 1-40 (1980).

Tong, David W., Eckart Walther and Kevin C. Zalondek. "Diagnosing an Analog Feedback System Using Model-Based Reasoning," *Preceedings of the AI Systems in Government Conference.* 290-295 Washington, D.C., 1989.

Tong, David W., Christopher H. Jolly and Kevin C. Salondek. *Diagnostic Tree Design with Model-Based Reasoning.* Report. Corporate Research and Development, General Electric Company, Schenectady, N.Y., 1989.

Wagner, R. E. and A.N. Blasdel. "An Approach to Autonomous Attitude Control for Spacecraft," *Proceedings of the Annual Rocky Mountain Guidance and Control Conference.* 51-64. Keystone, Colorado, February 1988.

Yost, Capt Raymond E. *Application of Model-Based Reasoning to Diagnosis of Faults in Inertial Navigation Equipment.* MS thesis, AFIT/GCS/ENG/89D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

Vita

Ralph W. Dries was born on the 29th May 1949 in Neuss, West Germany. In 1954 he emigrated, with his parents, to Australia. In 1966, he enlisted into the Royal Australian Air Force (RAAF) to undertake training as an apprentice radio/electronic technician. In 1978, under the RAAF's Civil Schooling Scheme, he undertook four years of under-graduate study at Royal Melbourne Institute of Technology (RMIT) and graduated in 1981 with a Bachelor of Engineering degree in electronic engineering. At RMIT he won the Rockwell International Fellowship and spent his first year after graduation (1982) at Rockwell's plant in Richardson Texas, USA, working as a civilian engineer designing military and civilian communication equipment. On return to Australia in 1983, he took on a systems engineering assignment on strategic communications systems at Headquarters Support Command in Melbourne, Victoria. In 1986, he was posted to No 3 Telecommunication Unit in Perth, Western Australia where he headed an engineering R&D section until May 1989, when he entered the School of Engineering, Air Force Institute of Technology at Wright-Patterson Air Force Base, Ohio, USA.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December, 1990	MS Thesis

4. TITLE AND SUBTITLE

Model-Based Reasoning in the Detection of Satellite Anomalies

5. FUNDING NUMBERS

6. AUTHOR(S)

Ralph W. Dries, B.E., FLTLT, Royal Australian Air Force

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GSO/ENG/90D-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Automatic fault detection and recovery would be a mandatory requirement for a satellite where some degree of autonomy is required. This thesis reviews some AI techniques used for the detection of satellite anomalies, and concludes that the model-based reasoning paradigm is best suited for automated on-board fault detection because it can cope with situations not necessarily programmed into the knowledge base. Using the Scheme language and its SCOOPS object oriented extension, development of software is described that models the pitch control channel in the attitude and velocity control subsystem of a typical geo-stationary communications satellite. This model is used by the model-based reasoning algorithm to diagnose faults in the real system. The algorithm used, is based on Scarl's "Full Consistency Algorithm", which is suitable for systems that have many sensors, but has limitations when applied to systems that are dependent on time or have feedback loops. These limitations were overcome by using a model that did not include time dependent objects and by "breaking the loop". It was found, for this problem domain, that the reasoner's model did not have to be identical to the real system to be able to successfully detect the cause of an anomaly.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Model-Based Reasoning, Deep Reasoning, Expert Systems, Artificial Intelligence, Diagnostics, Anomaly Resolution, Anomaly Detection, Satellites, Spacecraft	218
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL